Snake Wrangling For Kids

Learning to Program with Python

Linux Edition

Written by Jason R. Briggs

Snake Wrangling for Kids, Learning to Program with Python
by Jason R. Briggs

Version 0.7.7

Copyright ©2007.

Published by... ah, no one actually.

Cover art and illustrations by Nuthapitol C.

Website:
http://www.briggs.net.nz/log/writing/snake-wrangling-for-kids

Thanks To:
Guido van Rossum (for benevelont dictatorship of the Python language), the members of the Edu-Sig mailing list (for helpful advice and commentary), author David Brin (the original instigator of this book), Michel Weinachter (for providing better quality versions of the illustrations), and various people for providing feedback and errata, including: Paulo J. S. Silva, Tom Pohl, Janet Lathan, Martin Schimmels, and Mike Cariaso (among others). Anyone left off this list, who shouldn't have been, is en-tirely due to premature senility on the part of the author.

License:

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 New Zealand License. To view a copy of this license, visit *http://creativecommons.org/licenses/by-nc-sa/3.0/nz/* or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA. Below is a summary of the license.

You are free:

- to Share to copy, distribute and transmit the work
- to Remix to adapt the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

For any reuse or distribution, you must make clear to others the license terms of this work.

Any of the above conditions can be waived if you get permission from the copyright holder.

Nothing in this license impairs or restricts the author's moral rights.

python
powered

বাবুরাম সাপুড়ে,
কোথা যাস বাপুরে
আয় বাবা দেখে যা,
দুটো সাপ রেখে যা

পাইথন প্রোগ্রামিং (লিনাক্স সংস্করণ)

মূল: জ্যাসন আর ব্রিগস
অনুবাদ: অঙ্কুর আইসিটি ডেভেলপমেন্ট ফাউন্ডেশন

নির্দেশনায় :
মাহে আলম খান
খালেদা ইয়াসমিন ইতি

অনুবাদে :
আবু আশরাফ মাসনুন
আসমা জাহান মুক্তা
মুশফিকুর রহমান
রবিন মেহ্দী
শেখর কুমার বিশ্বাস

সার্বিক সম্পাদনা , বিন্যাস ও সমন্বয় :
আকলিমা শরমিন

বাবা-মা' কে বলছি... ... ...

প্রিয় মা/ বাবা/ অভিভাবক,
আপনার সন্তানকে আমরা নতুন একটি জগতের সাথে পরিচয় করিয়ে দিতে যাচ্ছি। তাকে প্রোগ্রামিং সম্পর্কে জানাতে চাচ্ছি। আমাদের দৃঢ় বিশ্বাস আপনি এতে আনন্দিত হবেন। তার জন্য আপনাকে একটি কাজ করতে হবে আর সেটি হচ্ছে আপনার কম্পিউটারে পাইথন ইনস্টল করতে হবে। আমাদের রচিত বইটি পাইথন ৩.০ সংস্করণ এর জন্য লেখা। পাইথনের এই সংস্করণটি পূর্ববর্তী সংস্করণগুলো থেকে একটু আলাদা। কাজেই ইনস্টল করার সময় এর সংস্করণ দেখে নিতে হবে।

পাইথন ইনস্টল করা তেমন জটিল কোন কাজ নয়। শুধু কোন ধরনের অপারেটিং সিস্টেম আপনি ব্যবহার করেন তার উপর নির্ভর করে কিছু কাজ করতে হবে। যদি এমন হয় যে আপনি মাত্রই সদ্য কেনা নতুন ঝকঝকে কম্পিউটারটি নিয়ে বাড়ি ফিরেছেন এবং সেটি দিয়ে কি করতে হবে তার বিন্দুমাত্র ধারণা নেই তাহলে নিশ্চয়ই পূর্ববর্তী বিবরণ শুনে আপনার মেরুদণ্ড বেয়ে ঠান্ডা স্রোতের ধারা বয়ে যাওয়ার মতো অনুভূতি হচ্ছে। একদম চিন্তা করবেন না। খুঁজে দেখুন আপনাকে সহায়তা করার জন্য আশেপাশে কাউকে না কাউকে পেয়ে যাবেন।

পাইথন ৩.০ (সর্বশেষ সংস্করণ) ইনস্টল করার জন্য তা ডাউনলোড করে নিতে হবে। আর এই কাজটি করতে আপনার কম্পিউটারের অবস্থা এবং ইন্টারনেট সংযোগের দ্রুততা আনুযায়ী ১৫মিনিট থেকে কয়েক ঘন্টা সময় লাগতে পারে। লিনাক্স-এর বিভিন্ন ডিস্ট্রিবিউশনের জন্য প্রতিটি ইনস্টলেশনের বিবরণ দেওয়া অনেক সময়ের ব্যাপার। তবে যদি আপনি লিনাক্স ব্যবহারকারী হয়ে থাকেন তাহলে ইতোমধ্যেই বুঝতে পারছেন কি করতে হবে। কাজেই ঝটপট আপনার কম্পিউটারে পাইথন ৩.০ ইনস্টল করে নিন।

ইনস্টলেশনের পর...
প্রথম কয়েকটি অধ্যায়ের জন্য আপনাকে অবশ্যই আপনার সন্তানের সাথে একটু সময় কাটাতে হবে। কিন্তু আমরা জানি কয়েকটি উদাহরণ পেয়ে যাবার পর সে কিবোর্ড থেকে আপনার হাত থেকে সরিয়ে নিজেই কাজ করতে চাইবে। পাইথন সম্পর্কে জানার পূর্বে একটু জেনে নিতে হবে কীভাবে টেক্সট এডিটর (Text Editor) ব্যবহার করতে হয়। না, মাইক্রোস্ফট ওয়ার্ডের মতো কোন ওয়ার্ড প্রসেসর সম্পর্কে তাদের জানতে হবে না। পুরোনো ধাঁচের কোন ওয়ার্ড টেক্সট এডিটর সম্পর্কে জানলেই চলবে। খুব সাধারণ কিছু ব্যবহার যেমন: ফাইল খোলা ও বন্ধ করা, নতুন টেক্সট ফাইল খোলা ও সংরক্ষণ করা ইত্যাদি জানতে হবে। আর আমাদের এই বইটি আসলে পাইথন সম্পর্কে খুব সহজ ভাষায় মৌলিক ধারণা দেয়ার চেষ্টা করেছে। আপনার সন্তান এই বইটি পড়ে স্বচ্ছন্দ অনুভব করবে এই নিশ্চয়তা আমরা দিতে পারি।

আপনার মূল্যবান সময় দেয়ার জন্য এবং সহযোগিতার জন্য আন্তরিক ধন্যবাদ।

সূচীপত্র

১. যে সাপের চোখ নেই, শিং নেই, নোখ নেই... ... ... ১

১.১ ভাষা নিয়ে কিছু কথা ২

১.২ নির্বিষ কিন্তু ভয়ংকর সরীসৃপ প্রাণী.. ... ... পাইথন ৩

১.৩ তোমার প্রথম পাইথন প্রোগ্রাম ৪

১.৪ তোমার দ্বিতীয় পাইথন প্রোগ্রাম... ... ... ৫

২. দুই গুণন দেড় সমান তিন... ... ... ৭

২.১ বন্ধনীর (Brackets) ব্যবহার এবং "অপারেশনের ধারাবাহিকতা" (Order of Operations) ৯

২.২ ভেরিয়েবল বা চলকের মতো অস্থির আর কিছুই নেই ১০

২.৩ ভেরিয়েবল বা চলকের ব্যবহার ১২

২.৪ এক টুকরো স্ট্রিং ১৩

২.৫ স্ট্রিংয়ের কিছু কৌশল ১৫

২.৬ স্পষ্ট একটি তালিকা তৈরি ১৬

২.৭ টাপল এবং তালিকা ১৯

২.৮ এসো নিজে করি ২১

৩. কাছিম এবং অন্যান্য ধীর গতির প্রাণী... ... ... ২৩

৩.১ তোমরা চেষ্টা কর ২৮

৪. পাইথনে প্রশ্ন করার পদ্ধতি ২৯

৪.১ এটি অথবা অন্যটি কর (Do this... or ELSE) !!! ৩০

৪.২ এটি করো.. অথবা ওটি করো... অথবা সেটি করো... অথবা অন্যটি করো!!! (Do this... or do this... or do this... or ELSE!!!) ৩১

৪.৩ দুটো শর্ত একসাথে ৩১

৪.৪ একাকীত্ব ৩২

৪.৫ পার্থক্য কী ...? ৩৩

৫. আবার এবং আবার ৩৫

৫.১ কখন একটি ব্লক বর্গক্ষেত্র নয় ? ৩৭

৫.২ যেহেতু আমরা লুপ নিয়ে আলোচনা করছি ৪২

৫.৩ নিজেরা একটু চেষ্টা করো ৪৫

৬. অব্যবহৃত? তাহলে নিশ্চয়ই পুনরায় ব্যবহারযোগ্য !!... ... ... ৪৬

৬.১ বিট ও পিস (Bits and Pieces) ৫০

৬.২ মডিউল (Modules) ৫১

৬.৩ এসো নিজে করি ৫৪

৭. ফাইল নিয়ে একটি ছোট অধ্যায় ৫৫

৮. কচ্ছপের মেলা **৫৭**
৮.১ রং করা ৬১
৮.২ অন্ধকার ৬৩
৮.৩ রং দিয়ে ভরাট করা ৬৪
৮.৪ নিজে করি ৭০

৯. একটুখানি গ্রাফিক্স **৭১**
৯.১ দ্রুত অঙ্কন ৭২
৯.২ সাধারণ অঙ্কন ৭৫
৯.৩ বক্স অঙ্কন করা ৭৬
৯.৪ বৃত্তের পরিধির অংশবিশেষ অঙ্কন করা (আর্ক) ৮০
৯.৫ উপবৃত্ত অঙ্কন করা ৮১
৯.৬ বহুভুজ অঙ্কন করা ৮৩
৯.৭ ইমেজ অঙ্কন করা ৮৪
৯.৮ মৌলিক অ্যানিমেশন ৮৬
৯.৯ রিচিং ইভেন্ট (Reacting to events. . .) ৮৮

১০. এখন আমরা কোথায় যাবো **৯০**

পরিশিষ্ট A পাইথনের মূল শব্দসমূহ **৯১**

পরিশিষ্ট B পূর্বনির্ধারিত ফাংশনগুলো **১০১**

পরিশিষ্ট C পাইথনের কিছু মডিউল **১০৮**

পরিশিষ্ট D ''এসো নিজে করি'' এর উত্তর **১১৬**

প্রথম অধ্যায়

যে সাপের চোখ নেই,
শিং নেই, নোখ নেই... ...

গতবারের জন্মদিনে পাঁপড়ী আন্টি তোমাকে একটি জামা উপহার দিয়েছিল। কিন্তু গায়ে দেবার পর আবিষ্কার করলে সেটি ঢোলা হয়েছে। মন এতটাই খারাপ হয়ে গেল যে তুমি ঠিক করলে তা আর কখনোই পরবে না। পাঁপড়ী আন্টি তাই এবারের জন্মদিনে তোমাকে কি উপহার দেবে তাই নিয়ে খুব চিন্তিত। এরমধ্যে একটি বই নিয়ে এক সহকর্মীর মন্তব্য তার কানে গেল। বইটি কম্পিউটারের কি যেন একটা বিষয় নিয়ে লেখা। মনে পড়ে গেল গত ছুটির দিনে অনেক আগ্রহ নিয়ে তুমি তাকে তোমার কম্পিউটারটি দেখিয়েছিলে এবং মজার মজার অনেক ব্যবহারও তাকে শিখিয়েছ। অবশেষে আন্টি তোমার জন্য সুন্দর এই বইটি নিয়ে গেলেন উপহার দেয়ার জন্য।

তোমরা কি আমার কথায় বিরক্ত হচ্ছো? আমি যদি এরচেয়ে রঙ্গিন কাগজে মোড়ানো কোন খেলনার কথা বলতাম তাহলে মনে হয় একটু বেশি খুশি হতে, তাই না? একেবারে শুরুতেই বললাম বই, তাও আবার কম্পিউটারের কি যেন নিয়ে লেখা !!! নিশ্চয়ই ভাবছো খুব খটমটে একটি বই তুমি পেতে যাচ্ছো। একটু অপেক্ষা করো, আস্তে আস্তে দেখাবো আমরা কি পেলাম। এক মুহূর্ত চিন্তা করে বলতো আমি আসলে কে। ধরো, তোমার বইয়ের তাকে রাখা জাদুকর নিয়ে লেখা একটি উপন্যাস হচ্ছি আমি। হয়তো আমার দাঁত আছে অথবা চোখ। আমার ভিতর এমন কিছু কল্পনা আছে ... ছবি আছে যা চলতে পারে। অথবা যখন তুমি আমার পৃষ্ঠা খুলবে তখন হয়তো আমি ভৌতিক শব্দ করবো...হু হা হা হা। কিন্তু সত্যি কি জানো.....আমার চোখ নেই।

আমার আছে সুন্দর ধারালো দাঁত.....

ভয় পেয়ে গেলে নাকি? ব্যাপারটা আসলে ততোটা খারাপ না। এমনকি আমি কথাও বলতে পারিনা, তুমি না তাকালে তোমার আঙ্গুলটাও কামড়ে দিতে পারবো না। আমি শুধু তোমার কম্পিউটারে কিছু কাজ করতে পারবো। ঐ যে বড় বড় ভারী যে জিনিসগুলো আছে সিপিইউ, মনিটর যেগুলো তার, কম্পিউটার চিপ, ক্যাবল এবং ডিভাইস দ্বারা যুক্ত থাকে সেগুলো নিয়ে নয় কিন্তু। তুমি নিশ্চয়ই জানো যে এগুলোর সঠিক স্থানে স্পর্শ না করে অন্য স্থানে বিশেষ করে বৈদ্যুতিক সংযোগের স্থানে স্পর্শ করলেই বৈদ্যুতিক শক খাবে (কাজেই ভুলেও সে স্থানগুলো স্পর্শ করো না)। এছাড়াও এমন কিছু জিনিস আছে যেগুলো তার, কম্পিউটার চিপ, ক্যাবল এবং ডিভাইসের মধ্যে লুক্কায়িত থাকে। যেগুলো থাকার ফলে কম্পিউটার আসলেই ব্যবহার উপযোগী হয় আমি সেই ধরনের কিছু কাজ করতে পারি।

তুমি নিশ্চয়ই এটি ভাববে না যে কম্পিউটার তো তোমার আছেই আর কিছুর কি দরকার। আচ্ছা ভেবে দেখ তো তুমি কি কাজ ছাড়া বসে থাকতে পারো? আমি জানি তুমি সবসময়ই কোন না কোন কাজ নিয়ে চিন্তা করো। আর যদি তা না হতো তাহলে ঘরে বসে শূণ্য দৃষ্টিতে ছাদের দিকে তাকিয়ে থাকতে। তোমার কম্পিউটারও তোমার মতো। চুপচাপ বসে থাকে পারে না। তাই কম্পিউটারকে নানান ধরনের কাজ দিতে হয়, তাকে দিয়ে নানান ধরনের কাজ করিয়ে নিতে হয়। এর জন্য কম্পিউটারে প্রোগ্রাম দরকার হয়। প্রোগ্রামবিহীন কম্পিউটার আর কর্মহীন তুমি দুটাই অর্থহীন।

এতক্ষণে আবিষ্কার করে ফেলেছ। সাবাস!

আমি শুধু একটা বই ছাড়া আর কিছুই না।

তোমাদের বাড়িতে নিশ্চয়ই সবাই মিলে খেলাধুলা করার জন্য অথবা বসার জন্য খোলা স্থান আছে। আর বিশেষ কোন অনুষ্ঠান বা প্রোগ্রাম ছাড়া নিশ্চয়ই সেখানে বসা হয়না। তুমি কি জানো, তোমার বাসার ডিভিডি প্লেয়ার, সম্ভবত ফ্রিজ, অথবা গাড়ি সবটাতেই কম্পিউটার প্রোগ্রাম আছে। প্রতিটি প্রোগ্রামই এসব বৈদ্যুতিক যন্ত্রপাতির কাজকে আরও বেশি কার্যকরী করে তুলতে সহায়তা করে। ডিভিডি প্লেয়ারে যে প্রোগ্রাম আছে তার কাজ হচ্ছে প্লেয়ারে কি চালাতে হবে তা বের করা। ফ্রিজের হয়তো একটি সাধারণ প্রোগ্রাম আছে যার কাজ কম বিদ্যুৎ ব্যবহারে কীভাবে খাবার ঠান্ডা রাখা। তোমার গাড়িতে হয়তো এমন একটি প্রোগ্রাম আছে যা গাড়ির সামনে কোন কিছু উঁচু দেখলেই সংকেত দিবে।

মজার ব্যাপার হচ্ছে, তুমি যদি কম্পিউটার প্রোগ্রাম লিখতে পারো তাহলে তুমি তোমার প্রয়োজনীয় সকল কাজই করতে পারবে। এমনও হতে পারে তুমি নিজের জন্য একটি গেম লিখে ফেললে, তোমার কাজের জন্য ওয়েব পেজ (web page) তৈরি করে ফেললে। আবার প্রোগ্রাম লিখতে পারলে এর সাহায্যে তোমার বাড়ীর কাজও করতে পারবে।

কি মনে হয়? একটি মজার জিনিস শিখতে যাচ্ছি, তাই তো?

১.১ ভাষা নিয়ে কিছু কথা

তোমার আম্মু যখন তোমাকে গল্প শোনায় তুমি কতো মনোযোগ দিয়েই না শোন! গল্পের মধ্যিখানে মাঝে মাঝে তুমি প্রশ্নও করো। অর্থাৎ তোমার আম্মু ও তুমি দুজনেই কথা বলতে পারো। আকাশে ওড়ে যে পাখিটি, পুকুরে সাঁতার কেটে বেড়ায় যে মাছটি অথবা বনে-বাদাড়ে ঘুরে বেড়ায় যে হরিণটি তারা প্রত্যেকেই কথা বলতে পারে। ওদের আম্মুরাও ওদেরকে অনেক গল্প শোনায়। যদিও শুনলে তুমি তা বুঝতে পারবেনা। কারণ, ওদের নিজস্ব ও আলাদা ভাষা আছে। ঠিক তোমনি কম্পিউটারেরও নিজের ভাষা আছে। কম্পিউটারও তার নিজের ভাষায় কথা বলতে ও বুঝতে পারে। আবার খেয়াল করো জাপানী শিশুরা জাপানী ভাষায় কথা বলে, ভারতীয় শিশুরা হিন্দী ভাষায় কথা বলে আর বাংলাদেশী শিশুরা বাংলা ভাষায় কথা বলে। অর্থাৎ মানুষ অনেক ধরনের ভাষা ব্যবহার করে কথা বলে। মজার ব্যাপার হচ্ছে কম্পিউটারও কথা বলার সময় কম্পিউটারের ভাষায় কথা বলে। আর মানুষের মতোই কম্পিউটারের রয়েছে একাধিক ভাষা যা ব্যবহার করে সে কথা বলতে পারে। কম্পিউটারে ব্যবহৃত ভাষার নাম প্রোগ্রামিং ভাষা (কাজেই কম্পিউটারও নিশ্চয়ই তার শিশু কম্পিউটারকে প্রোগ্রামিং ভাষায় গল্প বলে! হা হা হা) ।

কিছু কিছু প্রোগ্রামিং ভাষার নামকরণ মানুষের নামে হতে পারে। আবার Acronym বা নামের অদ্যাক্ষর নিয়ে গঠিত শব্দ অথবা টেলিভিশনের কোন অনুষ্ঠানের নামেও প্রোগ্রামিং ভাষার নামকরণ করতে দেখা যায়। এই নামগুলোরই কিছু বর্ণের পরে যদি তুমি যোগ চিহ্ন (+) এবং হ্যাস চিহ্ন (#) দাও তাহলে নামটির যে চেহারা দাঁড়াবে তাতে অন্য একটি প্রোগ্রামিং ভাষা তুমি পেতে পারো। ব্যাপারটিকে আরও জটিল করা যায় যদি এদের মধ্যকার কয়েকটি ভাষা প্রায় একই রকমের হয়। অর্থাৎ লেখার আদলে তেমন কোন পার্থক্যই যদি খুঁজে পাওয়া না যায়।

তাহলে কি দেখা গেল.....কোন ধারণা নেই !!

সৌভাগ্যক্রমে কম্পিউটারের এসব জটিল ভাষার কিছু কিছু একেবারেই অব্যবহৃত হয়ে পড়ছে অথবা পুরোটাই বিলুপ্ত হয়ে যাচ্ছে। যাই হোক, কম্পিউটারের সাথে কথা বলার অনেক রকম উপায় আছে। তার মধ্যে একটি উপায় নিয়ে আমরা এখন আলোচনা করতে চাই। বুঝতেই পারছ তা না হলে আমরা কখনোই কম্পিউটারের ভাষা বুঝতে পারবো না। ফলে মজার মজার গল্পগুলো থেকে বঞ্চিত হবো।

১.২ নির্বিষ কিন্তু ভয়ংকর সরীসৃপ প্রাণী

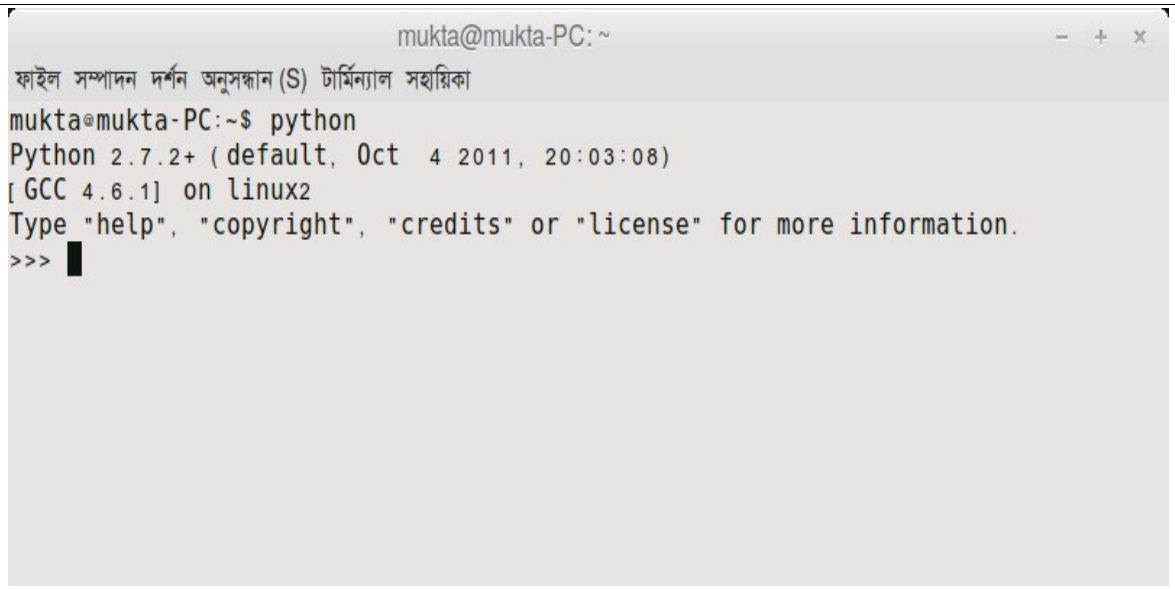
... ... ***পাইথন।***

ভয় পেয়ো না, আমি অজগর সাপের কথা বলছি না। আমি যে সাপের কথা বলছি তা কম্পিউটারের একটি প্রোগ্রামিং ভাষা। ও আচ্ছা, তুমি টেলিভিশনে প্রচারিত কোন অনুষ্ঠানের কথা ভাবছ? না তাও নয়। হয়তো তোমাদের জানা থাকবে মন্টি পাইথন নামে একটি ব্রিটিশ কমেডি শো আছে যা ১৯৭০ সাল থেকে এখনও পর্যন্ত জনপ্রিয় একটি টেলিভিশন অনুষ্ঠান হিসেবে প্রচারিত হয়ে আসছে। প্রোগ্রাম মানেই যে কোন টিভি চ্যানেল বা বন্ধু-বান্ধব নিয়ে ঘুরে বেড়ানোর কথা ভাবতে হবে সেই ধারণা কিন্তু এখন আর নেই। এর বাইরেও এক ধরনের প্রোগ্রাম এবং প্রোগ্রামিং ভাষা আছে। ঠিক ধরেছ, কম্পিউটারে ব্যবহৃত এক ধরনের ভাষা যার নাম দেয়া হয়েছে পাইথন।

কম্পিউটার প্রোগ্রাম শেখার সময় পাইথন সম্পর্কে জানা অত্যন্ত জরুরী। এর কারণ, যেই মুহূর্তে তুমি প্রোগ্রামিং শিখছ ঠিক সেই মুহূর্তেই দ্রুততার সাথে কাজ করার জন্য তোমার পাইথনের সাহায্য লাগবে।

হুমম.... এখনই তোমার আম্মু/আব্বু অথবা শিক্ষক যিনি তোমাকে কম্পিউটার শেখাচ্ছেন তাঁকে নিজ দায়িত্বে বইয়ের শুরুতে লেখা নোট "অভিভাবকের প্রতি....." পড়তে অনুরোধ করো। কারণ, এখন তোমার তাঁর সাহায্য লাগবে।

তাঁরা নোটটি ভালো করে পড়েছেন কিনা চলো যাচাই করে নেয়া যাক। প্রোগ্রামিং এর জন্য তোমাকে কোন টার্মিনাল



চিত্র ১.১ লিনাক্স-এর জন্য পাইথন কনসোল

অ্যাপ্লিকেশন ('Konsole', 'rxvt', xterm' অথবা ডজন খানেক প্রোগ্রাম থেকে যে কোন একটি) ব্যবহার করতে হবে এবং কেন সেটা নির্বাচন করা হলো তা জিজ্ঞাসা করলেই জানা যাবে তাঁরা কতোটা মনোযোগের সাথে নোটটি পড়েছেন। একটু মজা করে নিই, তাই না?

টার্মিনাল প্রোগ্রামটি চালু করে টাইপ করো 'python' (অবশ্যই উদ্ধৃতি চিহ্ন ব্যতীত) এবং এন্টার চাপো। চিত্র ১.১ এর মতো কিছু একটা দেখতে পাবে।

উপরের নির্দেশনা অনুসরণ করতে গিয়ে যদি এমন হয় যে তুমি কিছু একটা খুঁজে পাচ্ছ না তাহলে নিশ্চয়ই তোমার আম্মু/আব্বু নোটটি মনযোগ দিয়ে পড়েননি অথবা কাজ করার সময় কোথাও না কোথাও ভুল করেছেন। এখন তুমি কি করবে জানো? তাদের জন্য লেখা নোটটি আবার খোল এবং তাদের আবার পড়তে অনুরোধ করো। হতে পারে তিনি তখন পত্রিকা পড়ছেন অথবা টেলিভিশনে খবর দেখছেন তাতে কিছু যায় আসে না। তুমি বার বার অনুরোধ করায় দুটি ব্যাপার ঘটতে পারে। এক. ভুল হওয়ায় লজ্জা পেতে পারেন, দুই. তোমার ব্যবহারে বিরক্ত হতে পারেন। ব্যাপার যেটাই ঘটুক না কেন ফলাফল একই। তিনি পুনরায় ভালোভাবে পড়ে নির্দেশনা অনুযায়ী কাজ করবেন। অবশ্য ইচ্ছে করলে তুমি নিজেই ঐ অংশটুকু পড়ে নির্দেশনা অনুযায়ী পাইথন ইনস্টল করে নিতে পারো।

১.৩ তোমার প্রথম পাইথন প্রোগ্রাম

যেভাবেই হোক না কেন আমরা ধরে নিতে পারি অবশেষে তুমি পাইথন কনসোল চালু করতে পেরেছ। পাইথন কনসোল হচ্ছে পাইথনের কমান্ড এবং প্রোগ্রাম চালু করার একটা উপায়। যখন তুমি কনসোল চালু করবে (অথবা কমান্ড দেবে) তুমি একটি 'প্রম্পট' (prompt) পাবে। পাইথন কনসোলে 'প্রম্পট' হচ্ছে পরপর তিনটি তীরের মাথা অথবা তিনটি বৃহত্তর চিহ্ন (>)।
ফলে এটি দেখতে যেমন হয়:

```
>>>
```

পাইথন প্রোগ্রামে একই সাথে বেশ কয়েকটি কমান্ড দিলে কনসোল ছাড়াও অন্যান্য প্রোগ্রাম চালানো যাবে। কিন্তু এই মুহূর্তে আমরা কাজটিকে একেবারেই সহজ ও সাধারণ রাখতে চাই। তাই সরাসরি কনসোলের প্রম্পট (>>>) এ টাইপ করো। তাহলে নিচের বাক্যটি দিয়ে শুরু করা যাক, কি বলো?

```
print("Hello World")
```

উপরের কমান্ডটি লেখার সময় অবশ্যই উক্তি চিহ্ন ("") দিতে হবে। লাইনটি শেষ হলে এন্টার কী চাপ। আশা করছি নিম্নোক্ত লেখাগুলো দেখা যাবে:

```
>>> print("Hello World")
Hello World
```

প্রাপ্ত ফলাফলে প্রম্পট আবারও দৃশ্যমান। এর অর্থ হচ্ছে পাইথন কনসোল আরও কমান্ড নেবার জন্য প্রস্তুত।

অভিনন্দন!! জীবনের প্রথম পাইথন প্রোগ্রামটি তুমি কিন্তু ইতোমধ্যে লিখে ফেললে। এখানে প্রিন্ট (print) হচ্ছে এক ধরনের ফাংশন। এর অর্থ বন্ধনীর মধ্যে যাই লেখা হোক না কেন পরবর্তীতে তা আবারও ব্যবহৃত হবে।

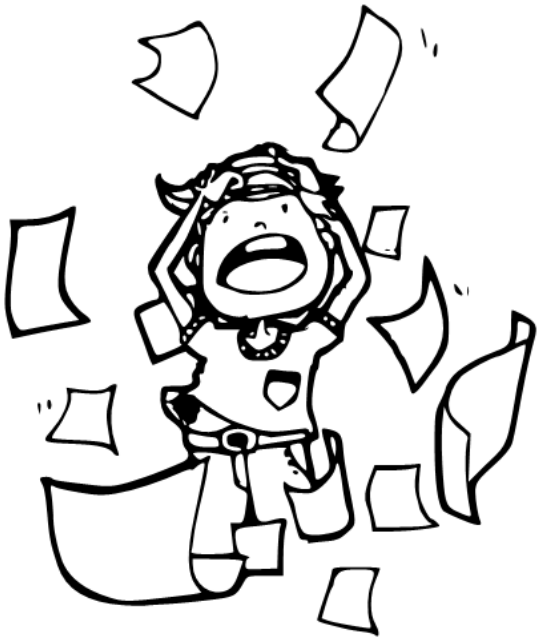
১.৪ তোমার দ্বিতীয় পাইথন প্রোগ্রাম... ... ...

ভাবছ, আবারও একই জিনিস?

কাজ করার সময় প্রতিবার যদি তোমাকে একই কমান্ড বার বার লিখতে হয় তাহলে আমরা যতটা ভাবছি পাইথন প্রোগ্রাম আসলে ততোটা উপকারী হবে না। আবার যদি এমন হয় যে কারও জন্য তুমি একটি প্রোগ্রাম লিখেছ কিন্তু ব্যবহার করার পূর্বে তাকে পুনরায় সেটি লিখতে হচ্ছে তাহলেও পাইথন প্রোগ্রাম লেখার কোন মানে হয় না।

তোমার কম্পিউটারের ওয়ার্ড প্রসেসর অ্যাপ্লিকেশনটি ব্যবহার করে নিশ্চয়ই ইতোমধ্যে অনেক অনেক অ্যাসাইনমেন্ট বা কাজ তুমি করেছো। তাহলে তোমার ওয়ার্ড প্রসেসর হচ্ছে এমন একটি জায়গা যেখানে ১০ থেকে ১০০ মিলিয়ন পর্যন্ত লাইন লেখা আছে। আচ্ছা ধরো, ওয়ার্ড প্রসেসরের অনেকগুলো লেখা তুমি প্রিন্ট করবে। এক একটা পৃষ্ঠায় কটা লাইন থাকবে এবং একদিকে না উভয়দিকে প্রিন্ট করা হবে তার উপর নির্ভর করে এইসব নথির স্তূপ কতটা বিশাল আকারে জমা হবে একটু হিসেব করো তো। হতে পারে চার লাখ পৃষ্ঠা... ... পাঁচ লাখ পৃষ্ঠা। কি বিশাল স্তূপ!!! শুধু এইটুকু কল্পনা করো সফটওয়্যারটি দোকান থেকে কিনে বাসায় আনতে তোমাকে যে ওজন বইতে হয়েছিল এখন সেই সফ্টওয়্যারে করা কাজের ওজন তার থেকেও কত গুণ বেশি। ভেবেই খুব মজা পাচ্ছি!!

... . এবং এই কাগজগুলো বয়ে নিয়ে যাওয়ার সময় যদি জোরে বাতাস বয়ে যায় তাহলে তো কথাই নেই। ধপাস! ভাগ্যিস একই নথি বার বার ব্যবহার করার জন্য প্রিন্ট বা টাইপিং এর বিকল্প আছে। তা না হলে মানুষ কোন কাজই করতে পারত না।



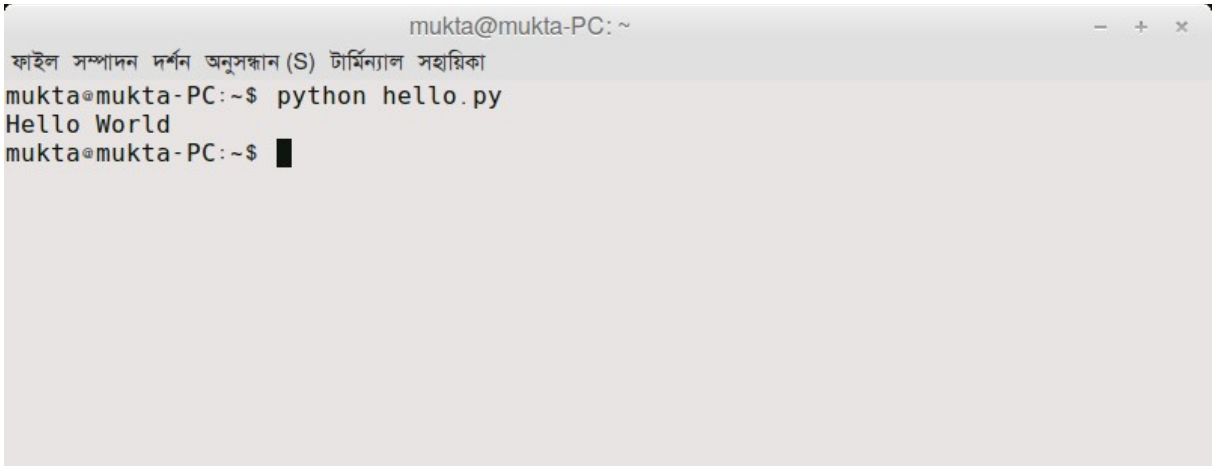
একটি নথি সম্পাদক (Text Editor) খোল। এখন আবারও আম্মু অথবা আব্বুকে একটু জ্বালাতন করো। অর্থাৎ জিজ্ঞেস করো কোন নথি সম্পাদক (Text Editor) ব্যবহার করবে। অতঃপর কনসোলে পূর্বলিখিত প্রিন্ট কমান্ডটি হুবহু লিখ:

```
print("Hello World")
```

File মেনুতে ক্লিক করে Save অপশনে যাও। সংরক্ষণ করার সময় যখন নথির নাম চাইবে তখন hello.py নামে Home folder এ সংরক্ষণ করো। এক্ষেত্রে সংরক্ষণের জন্য প্রদর্শনকৃত ডায়ালগ বক্স এ 'Home' এর জন্য একটি আইকন থাকবে। অতঃপর আগের মতো টার্মিনাল অ্যাপ্লিকেশনটি (Konsole, rxvt যেটা পূর্বে ব্যবহার করেছ) খুলে টাইপ করো:

```
python hello.py
```

পাইথন কনসোলে উপরের কমান্ড লেখার সাথে সাথে দেখা যাবে লেখা হয়েছে Hello World। চিত্র ১.২ দেখ।



চিত্র ১.২ লিনাক্স এ টেক্সট ফাইল থেকে পাইথন প্রোগ্রাম চালু করা

যে মানুষগুলো পাইথন আবিষ্কার করেছেন তারা এতটাই আমাদের কথা ভেবেছেন যে বার বার যেন একই জিনিস লিখতে না হয় তার ব্যবস্থা করেছেন। তোমার কাজকে তারা অনেক সহজ করে দিয়েছেন। অথচ এই মানুষগুলোই ১৯৮০ সালের দিকে কত কষ্ট করেই না কাজগুলো করতো। না, আমি একটুও বানিয়ে বলছি না। তোমার আব্বু-আম্মুকে অথবা তোমার শিক্ষককে জিজ্ঞাসা করে দেখো তারাও করতেন। তাদের কাজ করার পদ্ধতি দেখলে তো হাসতে হাসতেই তোমার পেটে খিল ধরে যাবে।

হাতে খড়ি হয়ে গেল

সবশেষে প্রোগ্রামিং এর মজার দুনিয়ায় তোমাকে স্বাগতম। আমরা খুব সাধারণ একটা অ্যাপ্লিকেশন দিয়ে শুরু করেছিলাম। প্রোগ্রামিং শেখার সময় সবাই তাই করে থাকে। পরবর্তী অধ্যায়ে আমরা পাইথন কনসোল ব্যবহার করে কিছু প্রয়োজনীয় কাজ করবো এবং দেখবো কিভাবে একটি প্রোগ্রাম তৈরি করতে হয়।

দ্বিতীয় অধ্যায়

দুই গুণন দেড় সমান তিন...

তোমরা কি বলতে পারো ৮ এর সাথে ৩.৫৭ গুণ করলে কত হয়? কেউ যদি তোমাকে এই গুণটি করতে বলে তাহলে তুমি কি করবে? নিশ্চয় ক্যালকুলেটর ব্যবহার করে গুণ করার চেষ্টা করবে কিংবা তুমি যদি হিসাবে অনেক ভালো হও অর্থাৎ মনে মনে অনেক বড় গুণ করতে পারো তবে সেই ভাবে চেষ্টা করে গুণফলটি বলতে চাইবে তবে এই রকমের কাজগুলো অনায়াসে পাইথন কনসোল ব্যবহার করে করা যায়। এখন আমরা পুনরায় কনসোল চালু করবো (কোন কারণে এড়িয়ে আসলে কনসোল চালু করা সম্পর্কে জানতে প্রথম অধ্যায় দেখ), কনসোলে একটি কমান্ড প্রম্পট দেখতে পাবে। কমান্ড প্রম্পটে 8 * 3.57 টাইপ করে কিবোর্ড থেকে এন্টার কী চাপো:

```
Python 3.0 (r30:67503, Dec 6 2008, 23:22:48)
Type "help", "copyright", "credits" or "license" for more
information.
>>> 8 * 3.57
28.559999999999999
```

পাইথন কনসোলে গুণ করার জন্য ব্যবহার করা হয় * (তারকা) চিহ্নটি (কিছু কিছু কিবোর্ডে এটি পাওয়া যাবে shift 8 কী চাপলে) সাধারণত আমরা স্কুলের গণিতের হিসাব করার সময় গুণ চিহ্ন বোঝাতে X ব্যবহার করে থাকি। কিন্তু পাইথনে X এর বদলে * ব্যবহার করতে হয় (* এর বদলে X ব্যবহার করলে কম্পিউটার বুঝতে পারবে না তুমি কি করতে চাইছো, গুণ না ইংরেজি হরফ X লিখতে চাইছো)। এখন আমরা দেখবো একটা সমীকরণে কীভাবে এটি বেশি দরকারী?

ধরা যাক, তোমার বাবা তোমাকে প্রতি মাসে স্কুলের টিফিনের টাকা হিসেবে ২০০ টাকা দেয়। এছাড়া তুমি প্রতি মাসে সরকার থেকে বৃত্তি পাও ৩০০ টাকা। তাহলে বছরে তুমি কত টাকা পাও। যদি এই হিসাবটি খাতায় লিখতে বলি তবে কি করবে? তুমি হয়তো লিখবে:
(৩০০+২০০) X ১২

পাইথন কি তবে কাজ করে না !!!!

তুমি যদি একটি ক্যালকুলেটরে 8X3.57 লেখ তবে এর উত্তর দেখতে পাবে:
28.56

তাহলে পাইথনে কেন হয় না? তবে কি পাইথন কাজ করেনা?

আসলে তা না। এর কারণ হলো কম্পিউটার ভগ্নাংশ (দশমিক সংখ্যার ভগ্নাংশ) নিয়ে একটু অন্যভাবে কাজ করে। এক্ষেত্রে ভগ্নাংশকে বলা হয় ফ্লোটিং পয়েন্ট বা ফ্লোট নম্বর (float number)। যারা প্রথম পাইথন নিয়ে কাজ শুরু করেছে তাদের জন্য এটি একটু জটিল সমস্যা। তাই সহজে মনে রাখার জন্য আমাদের মনে রাখতে হবে, যখন আমরা দশমিকের ভগ্নাংশ নিয়ে কাজ করবো অর্থাৎ যোগ-বিয়োগ, গুণ-ভাগ করবো তখন সঠিক ফলাফল নাও আসতে পারে।

যেখানে ২০০ এর সাথে ৩০০ যোগ করে এক বছরের ১২ মাস দিয়ে গুণ করার কথা বলা হয়েছে। তুমি যেহেতু হিসাবে পারদর্শি তাই জানো যে (২০০+৩০০) সমান ৫০০, তাই তোমার সমীকরণটি হবে:

৫০০ X ১২

কাজটি যেকোন ক্যালকুলেটরের জন্য অথবা একটি খাতায় লিখে করা খুবই সহজ। কিন্তু হিসাবটি আমরা কনসোলে এই ভাবে করতে পারি:

```
>>>(200+300)* 12
6000
>>>500* 12
6000
```

ধরো, যদি তুমি মাসে ২৫০ টাকা খরচ কর তবে কি হবে? বছর শেষে তোমার হাতে কত থাকবে? এই সমীকরণটি আমরা খাতায় ভিন্ন ভাবেও লিখতে পারি। চলো আমরা সমীকরণটি কনসোল এ লিখি:

```
>>> (200 + 300 - 250) * 12
3000
```

অর্থাৎ ২০০ এবং ৩০০ থেকে ২৫০ বিয়োগ করে এক বছরের ১২ মাস দিয়ে গুণ করতে হবে এবং তোমার হাতে বছর শেষে থাকবে ৩০০০ টাকা। এই কাজগুলো আমরা ক্যালকুলেটর ব্যবহার করে করতে পারি। কিন্তু এখন আমরা দেখবো কনসোলে কীভাবে কাজগুলো আরও সহজে করা যায়।

পাইথন কনসোলে আমরা গুণ, ভাগ (অবশ্যই ভাগও করা যাবে), যোগ, বিয়োগ করতে পারবো। এগুলো আমরা একই সাথে করতে পারবো, আবার ইচ্ছে করলে একসাথে অনেকগুলো গাণিতিক হিসেবও করতে পারবো কিন্তু আমরা এখন এখানে সেসব দেখবো না। এখন আমরা পাইথনের কিছু মৌলিক গাণিতিক চিহ্নের সাথে পরিচিত হবো (প্রকৃতপক্ষে চিহ্নগুলো অপারেটর নামে পরিচিত) চিহ্নগুলো হলো-

+	Addition/ যোগ
-	Subtraction/ বিয়োগ
*	Multiplication/ গুণ
/	Division/ ভাগ

এখানে লক্ষ কর যে, ভাগের চিহ্ন হিসাবে ফরোয়ার্ড স্ল্যাশ (/) ব্যবহার করা হয়েছে কারণ সমীকরণে ভাগের চিহ্ন (÷) ব্যবহার করাটা জটিল। (যেহেতু কম্পিউটারের কিবোর্ড দিয়ে ÷ চিহ্নটি সহজে লেখা যায় না)। উদাহরণ হিসাবে বলা যায়, তোমার যদি ১০০ টি ডিম থাকে এবং ২০ টি বাক্স থাকে তবে তুমি জানতে চাইতে পারো প্রতিটি বাক্সে কতটি ডিম আছে, এজন্য তুমি ১০০ কে ২০ দিয়ে ভাগ করবে আর এই জন্য সমীকরণে তুমি লিখবে :

১০০

২০

অথবা তুমি যদি ভাগ প্রক্রিয়া সম্পর্কে জান তবে তুমি লিখতে পারো:

```
     ৫
    -----
২০)১০০
    ১০০
----------
       ০
```

অথবা তুমি এভাবে লিখতে পারো: ১০০ ÷ ২০
কিন্তু পাইথনে এ জন্য তোমাকে লিখতে হবে "100 / 20"।

আমি মনে করি এটি অনেক সহজ, কিন্তু আমি যেহেতু একটি বই- তাই তুমি আমার চেয়ে ভালো জানো।

২.১ বন্ধনীর (Brackets) ব্যবহার এবং "অপারেশনের ধারাবাহিকতা " (Order of Operations)

আমরা প্রোগ্রামিং-এর ভাষায় বিভিন্ন নিয়ন্ত্রণের কাজ করে থাকি বন্ধনী দ্বারা একে বলা হয় "অপারেশনের ধারাবাহিকতা"। যেকোন অপারেটরের ব্যবহারকে বলা হয় অপারেশন (উপরের টেবিলে দেখানো প্রতিটি চিহ্নই এক একটি অপারেটর)। এগুলো ছাড়াও অনেক ধরনের অপারেটর আছে কিন্তু তালিকায় দেয়া অপারেটর (যোগ, বিয়োগ, গুণ, ভাগ) তুলনামূলক ভাবে অনেক সহজ। এখন আমাদের শুধু এইটুকু জানলেই চলবে গুণ-ভাগের কাজ, যোগ-বিয়োগের কাজের চেয়ে আগে করতে হয়। যদি সব অপারেটর যোগের (+) হয় তবে ধারাবাহিকতা হবে:

```
>>> print(5 + 30 + 20)
55
```

যদি একই সমীকণের যোগ এবং বিয়োগ করার অপারেটর থাকে তবে পাইথনে এটি সম্পাদন করা হবে নিচের মতো করে:

```
>>> print(5 + 30 - 20)
15
```

কিন্তু নিচের সমীকরণে আমরা দেখতে পাই, এখানে একটি গুণের অপারেটর আছে বলে পাইথন প্রথমে ২০ ও ৩০ কে বিবেচনা করবে, সমীকরণকে যদি আমরা কথায় লিখি তাহলে লিখতে হবে, ৩০ কে ২০ দিয়ে গুণ করে তার সাথে ৫ যোগ করলে কত হয় (গুণের কাজ আগে করতে হবে কারণ ধারাবাহিকতায় গুণের অবস্থান যোগের উপরে) অর্থাৎ পাইথন কোড হবে:

```
>>> print(5 + 30 * 20)
605
```

এখন আমরা যদি একই সমীকরণে যদি একটি বন্ধনী ব্যবহার করি তাহলে কি হবে? নিচের সমীকরণের ফলাফলটি দেখ:

```
>>> print((5 + 30) * 20)
700
```

ফলাফলে এই পার্থক্য কেন? কারণ বন্ধনী এই ধারাবাহিকতা নিয়ন্ত্রণ করছে। বন্ধনী থাকলে পাইথন তার কাজ আগে করে, এরপর করে অন্য অপারেটরের কাজ। যেমন আগের সমীকরণটিকে আমরা বলতে পারি, "৫ এর সাথে ৩০ যোগ করে তাকে ২০ দ্বারা গুণ করা।" একটি সমীকরণে যদি একাধিক বন্ধনী থাকে তবে তার হিসাব করা আরো বেশি জটিল। যেমন নিচের সমীকরণে বন্ধনীর ভেতর বন্ধনী ব্যবহার করা হয়েছে:

```
>>> print(((5 + 30) * 20) / 10)
70
```

এক্ষেত্রে, পাইথন প্রথমে সবচেয়ে ভেতরের বন্ধনীর কাজ আগে করবে, এরপর ধীরে ধীরে করে বাইরের বন্ধনীর কাজ এবং অন্যান্য অপারেটরের কাজ করবে। সুতরাং আমরা এই সমীকরণকে বলতে পারি "৫ এর সাথে ৩০ যোগ করে তার সাথে ২০ গুণ করে গুণফলকে ১০ দিয়ে ভাগ করা"। বন্ধনী ছাড়া ফলাফল কিছুটা ভিন্ন হবে, যেমন:

```
>>> 5 + 30 * 20 / 10
65
```

এক্ষেত্রে প্রথমে ৩০ এর সাথে ২০ গুণ করতে হবে, অতঃপর ফলাফলকে ভাগ করতে হবে ১০ দিয়ে, শেষে ফলাফলের সাথে ৫ যোগ করলে চূড়ান্ত ফলাফল পাওয়া যাবে।

মনে রাখতে হবে যে, যতক্ষণ পর্যন্ত বন্ধনী ব্যবহার করা না হবে ততক্ষণ গুণ এবং ভাগের কাজ সবসময়ই যোগ এবং বিয়োগের আগে হবে।

২.২ ভেরিয়েবল বা চলকের মতো অস্থির আর কিছুই নেই

প্রোগ্রামে বহুল ব্যবহৃত একটি শব্দ হচ্ছে চলক বা ভেরিয়েবল (variable)। ভেরিয়েবল ব্যবহার করা হয় কোন কিছু সংরক্ষণের স্থানের বর্ণনা করতে। ভেরিয়েবল হিসাবে যে কোন সংখ্যা, অক্ষর বা সংখ্যার তালিকা অথবা শব্দ অথবা অন্য যে কোন আইটেম নির্ধারণ করা যেতে পারে।

যেমন আমরা যদি ভেরিয়েবলকে একটি ডাকবাক্স হিসাবে চিন্তা করি তবে এর ভেতরে চিঠি বা অন্যান্য কিছু রাখতে পারি। সেই ভাবে আমরা ভেরিয়েবল হিসাবে যে কোন কিছু নির্ধারণ করতে পারি। এই মেইল বক্স আইডিয়া অনেক প্রোগ্রামিং ভাষাই আছে কিন্তু সব প্রোগ্রামে নেই।

তবে পাইথনে ভেরিয়েবলের ব্যাপারটি একটু আলাদা।
কোন কিছুকে ভেরিয়েবল হিসাবে ঘোষণা দিতে তার একটি নাম দিতে হয় ও ভেরিয়েবলের আগে একটি সমান চিহ্ন (=) দিতে হয়। ভেরিয়েবল তৈরির পর পাইথনকে বলে দিতে হয় আমরা এই ভেরিয়েবল দিয়ে কি কাজ করতে চাইছি, একটি উদাহরণ দিয়ে বিষয়টি বোঝানো হলো:

```
>>> ektara = 100
```

লক্ষ কর, আমরা একটি ভেরিয়েবল তৈরি করেছি যার নাম দিয়েছি ' ektara এবং বলে দিয়েছি এটি একটি সংখ্যা

ধারণ করবে যার মান ১০০। যাতে পরবর্তীতে আমরা চাইলেই পাইথন এই মান বলে দিতে পারে তাই এই মানটি মনে রাখার জন্য পাইথনকে বলে দেওয়া হয়। আমরা যদি পরবর্তীতে এই মানটি খুঁজে পেতে চাই তবে কনসোলে শুধুমাত্র 'print' কমান্ড লিখে প্রথম বন্ধনীর মাঝে ভেরিয়েবলের নাম লিখে কিবোর্ডের enter কী চাপলেই পাইথন ভেরিয়েবলের মানটি দেখাবে, যেমন:

```
>>> ektara = 100
>>> print(ektara)
100
```

যদি আমরা ভেরিয়েবলের মান পরিবর্তন করি তবে পাইথনও ফলাফলে সেই পরিবর্তিত মান দেখাবে, যেমন:

```
>>> ektara = 200
>>> print(ektara)
200
```

লক্ষ কর, প্রথম লাইনে আমরা বলে দিচ্ছি ektara ভেরিয়েবলটি ২০০ নির্দেশ করবে এবং দ্বিতীয় লাইনে আমরা জানতে চাচ্ছি ektara কি নির্দেশ করছে। তখন পাইথন ফলাফল হিসেবে পরিবর্তনটি দেখাচ্ছে। আমরা এভাবে ভেরিয়েবল হিসাবে একাধিক আইটেমকে নির্দেশ করতে পারি, যেমন:

```
>>> ektara = 200
>>> ankur = ektara
>>> print(ankur)
200
```

উপরের কোডে দেখতে পাই, আমরা একটি নাম ankur ঠিক করেছি যা ektara-কে নির্দেশ করে, আবার ektara নির্দেশ করে একটি সংখ্যা ২০০। এখানে লক্ষ কর, আমরা ektara এর মান হিসেবে যাই রাখিনা কেন সেই মানই পরবর্তীতে ektara প্রকাশ করবে। তাই আমাদের ভেরিয়েবলের নাম নির্ধারণের সময় একটু কৌশলী হতে হবে। এখানে আমরা ভেরিয়েবলের নাম হিসাবে ektara ব্যবহার করেছি। কিন্তু ektara একটি সাধারণ নাম। যা সচারচর ব্যবহার করা হয়।

আমরা এর আগে ডাকবাক্সের কথা বলেছিলাম, যেটা শুধু মাত্র একটি কাজেই অর্থাৎ ডাক বা চিঠি রাখার কাজেই ব্যবহার করা হয় কিন্তু একটি ভেরিয়েবল নানান ধরনের কাজের জন্য ব্যবহার করা হয়। তাই ভেরিয়েবলের নাম নির্ধারণ একটি গুরুত্বপূর্ণ বিষয়।

এ বিষয়ে একটি উদাহরণ দিলে বিষয়টি সম্পর্কে ধারণা অনেক স্পষ্ট হয়ে যাবে। মনে করো, তুমি পাইথন কনসোল চালু করে টাইপ করলে ‘ektara = 200’ অর্থাৎ একটি ভেরিয়েবল নির্ধারণ করলে ektara যার নাম নির্দেশ করলে ২০০। মানটি নির্দেশ করার পর তুমি তোমার বন্ধুদের সাথে হিমালয় জয় করতে বের হয়ে গেলে। হিমালয় জয় করে সাহারা মরুভূমিও জয় করার উদ্দেশ্যে রওয়ানা হলে, সাহারা মরুভূমি জয় করে তুমি ইংলিশ চ্যানেলও পার হয়ে আমাজন বন ঘুরে এসে, আমাজন নদীতে নৌকা চালিয়ে অনেক দূরে চলে গেলে। এইসব শেষ করে তুমি যখন বাসায় ফিরে তোমার কম্পিউটারের সামনে বসলে তখন কি তুমি মনে করতে পারবে এই ২০০ মানে কি বোঝাচ্ছে এবং এটা কেন লেখা হয়েছিল?

আমার মনে হয় না আমি মনে করতে পারবো।

আমি কিছুক্ষণ আগেই বলেছিলাম 'ektara=200' কিন্তু এখনই ভুলে গেছি এটা দ্বারা কি বোঝায়? তাহলে ১০ বছর পরে আমি কীভাবে মনে করতে পারবো বা তার সম্ভাবনাই কত?

কিন্তু আমি যদি এই ভেরিয়েবলের নাম এই ভাবে দেই number_of_students, অর্থাৎ আমরা যদি লিখি:

```
>>> number_of_students = 200
```

আমরা এভাবে লিখতে পারি কারণ, ভেরিয়েবলের নাম অক্ষর, সংখ্যা এবং আন্ডারস্কোর (-) সহকারে হতে পারে। শুধু মাত্র মনে রাখতে ভেরিয়েবলের নাম সংখ্যা দিয়ে শুরু করা যাবে না। তুমি যদি ১০ বছর পরেও ফিরে আসো তবে ভেরিয়েবলের নাম 'number of students' দ্বারা কি বোঝায় তা তুমি বলতে পারবে। তখন তুমি টাইপ করতে পারবে:

```
>>> print(number_of_students)
200
```

এবং তুমি বুঝতে পারবে এখানে ২০০ ছাত্রের সংখ্যার কথা বলা হচ্ছে। তবে ভেরিয়েবলের নাম সবসময়ই অর্থবহ নাও হতে পারে। এটা কখনও একটি মাত্র অক্ষরেও হতে পারে যেমন ('a'), আবার বড় কোন বাক্য ব্যবহার করেও করা যায়, অথবা তুমি যদি খুব দ্রুত কোন প্রোগ্রাম লিখতে চাও তবে কাজের সুবিধার জন্য খুব সাধারণ ও সহজ একটি নাম ও দিতে পারো। এটা নির্ভর করবে তুমি পরবর্তীতে কীভাবে এটিকে ব্যবহার করতে চাও ও তুমি কীভাবে চিন্তা করছো তার উপর।

ভেরিয়েবলের নাম দেওয়ার সময় মনে রাখতে হবে, একটি যৌক্তিক নামের ভেরিয়েবল সবসময় প্রয়োজনীয় ভেরিয়েবল নাও হতে পারে।

২.৩ ভেরিয়েবল বা চলকের ব্যবহার

আমরা জানি কীভাবে একটি ভেরিয়েবল তৈরি করতে হয় কিন্তু এটি কীভাবে ব্যবহার করবো? আমাদের কি সেই সমীকরণটির কথা মনে আছে? যেখানে বলা হয়েছিল, তুমি প্রতি মাসে টিফিনের জন্য পাও ২০০ টাকা আর বৃত্তির টাকা হিসেবে পাও ৩০০টাকা এবং মাসে তোমার খরচ হয় ২৫০টাকা, তাহলে বছর শেষে তোমার হাতে কত টাকা থাকবে, তাহলে সমীকরণটি হবে:

```
>>>((200+300) - 250)* 12
3000
```

আমরা যদি প্রথম তিনটি নম্বরকে ভেরিয়েবল হিসাবে কল্পনা করি তবে আমাদের লিখতে হবে:

```
>>> tiffin = 200
>>> stipend = 300
>>> spending = 250
```

এখানে আমরা তিনটি ভেরিয়েবল তৈরি করেছি এবং তাদের নাম দিয়েছি 'tiffin', 'stipend' and 'spending' । এখন আমরা সমীকরণটিকে নতুন করে লিখতে পারি:

```
>>> print((tiffin + stipend - spending) * 12)
3000
```

লক্ষ কর, ভেরিয়েবল তৈরি করায় আমরা এখন অতিরিক্ত কিছু কাজ করতে পারবো। যেমন ধরা যাক, তুমি যদি মাসে টিফিনের জন্য আরও 50 টাকা বেশি পাও তবে শুধু মাত্র ভেরিয়েবলের 200 এর স্থলে 250 লিখতে হবে। এজন্য আপ-অ্যারো কী (up-arrow key ↑) চাপতে হবে যতক্ষণ পর্যন্ত না tiffin=200 আসে। এখন 200 মুছে 250 লিখে এন্টার চাপলেই আমরা ফলাফল পাবো, যেমন:

```
>>> tiffin = 250
>>> print((tiffin + stipend - spending) * 12)
3600
```

এখন দেখ তোমার বাৎসরিক আয় বেড়ে হয়েছে 3600। এখন তুমি নিজে নিজে অন্যান্য ভেরিয়েবল পরিবর্তন করে দেখতে পারো, ফলাফল পরিবর্তন হয়ে যাচ্ছে। যেমন তুমি যদি তোমার মাসিক ব্যয় 50 টাকা বাড়িয়ে দাও, তাহলে তোমার বাৎসরিক আয় হবে:

```
>>> spending = 300
>>> print((tiffin + stipend - spending) * 12)
2400
```

অর্থাৎ বছর শেষে তোমার হাতে রইলো 2400 টাকা। এভাবে আমরা ভেরিয়েবলের একটি খুব সাধারণ ব্যবহার দেখলাম। এখনও আমরা ভেরিয়েবলের সত্যিকারের বড় কোন ব্যবহার দেখিনি। তবে এই ব্যবহার থেকেও আমরা বুঝতে পারি যে ভেরিয়েবল ব্যবহার করা হয় কোন কিছু সংরক্ষণ করার কাজে।

ভেরিয়েবলকে একটি লেভেলসহ ডাকবাক্স হিসাবেও মনে করতে পারো!

২.৪ এক টুকরো স্ট্রিং

তোমাদের নিশ্চয়ই মনে আছে বইটিতে আমরা আগেই ভেরিয়েবল সম্পর্কে বলেছি। আমরা জানি যে, ভেরিয়েবল হিসাবে শুধু সংখ্যাই নয় যে কোন শব্দও আমরা নির্ধারণ করে দিতে পারি। প্রোগ্রামে বেশির ভাগ ক্ষেত্রেই আমরা টেক্সটকে স্ট্রিং হিসাবে বিবেচনা করে থাকি। কিন্তু স্ট্রিং শুধু মাত্র শব্দ বা শব্দ গুচ্ছই না এ ছাড়াও আরো কিছু।

এবং আবারো বলছি , সম্ভবত এটি শুধুমাত্র স্ট্রিংই নয়।

এই ক্ষেত্রে, আমাদের জানতে হবে যে, একটি স্ট্রিং হচ্ছে কয়েকটি অক্ষর, নাম্বার এবং চিহ্নের সমষ্টি যা একসাথে একটি অর্থ প্রকাশ করে। যেমন এই বইয়ের সকল অক্ষর, সংখ্যা এবং চিহ্ন মিলে একটি অর্থ প্রকাশ করে। অথবা তোমার নাম ও একটি স্ট্রিং হতে পারে অথবা তোমার বাসার ঠিকানাও একটি স্ট্রিং হতে পারে। তোমাদের নিশ্চয়ই মনে আছে আমরা প্রথম অধ্যায়ে একটি পাইথন প্রোগ্রাম বানিয়েছিলাম এবং সেখানে একটি স্ট্রিং ব্যবহার করেছিলাম, যেটা ছিল: ‘Hello World’।

পাইথনে একটি শব্দকে উক্তি চিহ্ন (") ভেতর রেখে স্ট্রিং তৈরি করতে হয়। এখন আমরা আমাদের পূর্বে করা ektara ভেরিয়েবলে একটি স্ট্রিং রাখবো, নিচের উদাহরণটি দেখ:

```
>>> ektara = "this is a string"
```

আমরা এখন খুব সহজে দেখতে পারবো ektara ভেরিয়েবলের ভেতর কি রেখেছি, এ জন্য টাইপ করতে হবে:

```
>>> print(ektara)
this is a string
```

একক উক্তি চিহ্ন ব্যবহার করে একটি স্ট্রিং তৈরি করতে হয়, যেমন:

```
>>> ektara = ’this is yet another string’
>>> print(ektara)
this is yet another string
```

কিন্তু আমরা যদি টেক্সট হিসাবে একক উক্তি চিহ্ন (Single quote ' ') বা দ্বিরুক্তি চিহ্ন (Double quote “ ”) ভেতর এন্টার চেপে একাধিক লাইন লিখে থাকি তবে ত্রুটি দেখাবে, অর্থাৎ প্রথম লাইন লিখে কমা শেষ না করে এন্টার চাপ দিলে ত্রুটি দেখাবে। একটি উদাহরণ দিয়ে বিষয়টি আমরা বুঝতে পারি:

```
>>> ektara = "this is two
     File "<stdin>", line 1
          ektara = "this is two
                               ^
SyntaxError: EOL while scanning string literal
```

(আমরা ত্রুটি বিষয়ে পরবর্তীতে আরও জানবো)

একাধিক লাইন লিখতে তিনটি কমা ব্যবহার করতে হবে। যেমন:

```
>>> ektara = ’’’this is two
... lines of text in a single string’’’
```

আমরা print কমান্ড দিয়ে দেখাতে পারি যে তিনটি কমা কাজ করে:

```
>>> print(ektara)
this is two
lines of text in a single string
```

লক্ষ কর, তুমি যদি তিনটি কমা ব্যবহার করে থাকো তবে প্রথম লাইন লেখার পর এন্টার দেওয়ার পর তিনটি ডট (...) দেখায়। এভাবে যতবার এন্টার দিবে ততবার তিনটি ডট আসবে ও নতুন লাইন লিখতে পারবে। এই ডটের পর তুমি যে কোন কিছু লিখতে পারবে।

২.৫ স্ট্রিংয়ের কিছু কৌশল

আচ্ছা তোমাদের একটি মজার প্রশ্ন করি, বলতো, 10 * 5 (১০ বার ৫) এর উত্তর কত হবে? হ্যাঁ ঠিক ধরেছো, উত্তর হবে 50। তুমি বলতে পারো এটা কোন মজার প্রশ্ন হলো? কারণ তুমি এই অংক সহজেই করতে পারো।
আচ্ছা এখন যদি বলি 10 * 'a' (১০ বার a অক্ষর) এর উত্তর কি হবে? তখন তুমি হয়তো বলবে এটি একটি অর্থহীন প্রশ্ন। কিন্তু পাইথনের জগতে এর উত্তরও আছে, তুমি পাইথন কনসোলে লেখ >>> print(10 * 'a') এবং এন্টারে চাপ দিলে ফলাফল দেখবে:

```
>>> print(10 * 'a')
aaaaaaaaaa
```

এই কাজটি শুধু মাত্র একটি একক শব্দের ক্ষেত্রে হবে, নিচের উদাহরণটি দেখ:

```
>>> print(20 * 'abcd')
abcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdab
cdabcdabcdabcd
```

স্ট্রিং লেখার অন্য আরেকটি কৌশল হলো, স্ট্রিংয়ে মানকে স্থাপন (embed) করে দেওয়া। এই কাজটি করা যায় %s ব্যবহার করে যা একটি মার্কার হিসাবে ব্যবহৃত হয়। এই %s স্ট্রিং ব্যবহার করে পরবর্তীতে আমরা যে কোন মান যুক্ত করে দিতে পারি। একটি উদাহরণ দিয়ে বিষয়টি পরিষ্কার করে দেওয়া হলো:

```
>>> mytext = 'I am %s years old'
>>> print(mytext % 12)
I am 12 years old
```

প্রথম লাইনে কয়েকটি শব্দ ও একটি প্লেস হোল্ডার (placeholder-%s) ধারণ করে এমন একটি ভেরিয়েবল mytext তৈরি করা হয়েছে। এখানে ব্যবহৃত %s একটি সংকেত, যার অর্থ পাইথন কনসোলে "আমাকে কোন কিছু দিয়েপ্রতিস্থাপন কর" । এ কারণে পরবর্তী লাইনে আমরা যখন print(mytext) লিখলাম, তখন আমরা % চিহ্নটি ব্যবহার করলাম এবং পাইথনকে বললাম এই % এর পরিবর্তে 12 ব্যবহার করতে। এভাবে আমরা % এর পরিবর্তে অন্যান্য কিছু ব্যবহার করতে পারি। নিচের উদাহরণগুলো দেখ:

```
>>> mytext = 'Hello %s, how are you today?'
>>> name1 = 'Lipi'
>>> name2 = 'Rana'
>>> print(mytext % name1)
Hello Lipi, how are you today?

>>> print(mytext % name2)
Hello Rana, how are you today?
```

উপরের উদাহরণে আমরা দেখতে পাই, এখানে তিনটি ভেরিয়েবল (mytext, name1 and name2) তৈরি করা হয়েছে- প্রথমটিতে আমরা % মার্কাপ ব্যবহার করেছি। এখন আমরা 'mytext' ভেরিয়েবলটি print করতে পারি এবং পুনরায় % এর পরিবর্তে 'name1' এবং 'name2' ব্যবহার করতে পারি। এছাড়াও তুমি একাধিক প্লেস

হোল্ডার ব্যবহার করতে পারো, নিচের উদাহরণটি দেখ:

```
>>> mytext = 'Hello %s and %s, how are you today?'
>>> print(mytext % (name1, name2))
Hello Lipi and Rana, how are you today?
```

যখন একাধিক মার্কার ব্যবহার করা হবে তখন মার্কারগুলো বন্ধনীর মধ্যে রাখতে হবে। অর্থাৎ দুইটি মার্কার থাকলে তা লেখার সঠিক নিয়ম হলো- (name1, name2)। অনেকগুলো মান বা মার্কারকে আমরা যখন তৃতীয় বন্ধনীর মধ্যে রাখবো তখন তাকে টাপল (tuple) বলা হয়। এটি অনেকটা লিস্টের মতো। যার সম্পর্কে আমরা পরে জানবো।

২.৬ স্পষ্ট একটি তালিকা তৈরি

তুমি একটি তালিকা তৈরি করলে মাছ, মাংস, ডিম, আলু, দুধ, কলা কেনার জন্য। এটি একটি পরিপূর্ণ বাজার তালিকা নয়। কিন্তু আমাদের কেনার জন্য যথেষ্ট। আমরা যদি এগুলোকে একটি ভেরিয়েবলের স্ট্রিং হিসাবে সংরক্ষণ করতে চাই তবে তা হবে:

```
>>> shopping_list = 'eggs, milk, fish, banana, potato, meat'
>>> print(shopping_list)
eggs, milk, fish, banana, potato, meat
```

পাইথনে অন্য ভাবেও আমি একটি 'list' তৈরি করতে পারি, যেখানে আমি একটি অবজেক্ট ব্যবহার করবো, নিচের উদাহরণটি দেখ:

```
>>> shopping_list = [ 'eggs', 'milk', 'fish', 'banana', 'potato',
'meat' ]
>>> print(shopping_list)
[ 'eggs', 'milk', 'fish', 'banana', 'potato',  'meat' ]
```

এখানে আমাকে অনেক কিছু লিখতে হয়েছে, কিন্তু তা আমার জন্যই উপকারী। যেমন আমরা এই লিস্ট থেকে সহজেই তৃতীয় আইটেমটি (একে index position বলা হয়) print করতে পারি, যা [ ] বন্ধনীর ভেতর থাকে যেমন:

```
>>> print(shopping_list[2])
fish
```

লিস্ট শুরু হয় 0 নাম্বার পজিশন থেকে, একারণে প্রথম আইটেমের অবস্থান 0, এভাবে দ্বিতীয়টির পজিশন 1, তৃতীয়টির অবস্থান 2। এটা অধিকাংশ সাধারণ মানুষই বুঝতে পারেন না, তবে প্রোগ্রামারদের কথা আলাদা। যেমন, তুমি আর তোমার বোন একসাথে সিঁড়ি দিয়ে উঠতে গিয়ে সিঁড়ির ধাপ গুনতে থাকো। তুমি হয়তো গোনা শুরু করলে 0 থেকে তবে তোমার বোন এই কাজটি করতে গেলে গুলিয়ে ফেলবে। কারণ সাধারণত আমরা গুনে থাকি ১ থেকে। কিন্তু পাইথনে লিস্ট তৈরির সময় গোনা শুরু হবে 0 থেকে।
এখন আমরা আগের উদাহরণে, লিস্টে থাকা তৃতীয় অবস্থান থেকে শুরু করে ৫ম স্থানের সব আইটেম দেখাতে পারবো, শুধু মাত্র [ ] বন্ধনীর ভেতরে একটি কোলন ব্যবহার করে, যেমন:

```
>>> print(shopping_list[2:5])
['fish', 'banana', 'potato' ]
```

আমরা ইনডেক্সের ২য় থেকে ৫ম অবস্থানে জিনিসের তালিকা দেখতে চাই কথাটিকে আমরা [2:5] এভাবে লিখে প্রকাশ করতে পারি। তালিকা ব্যবহার করে সকল আইটেমগুলো শৃঙ্খলিত ভাবে সংরক্ষণ করার জন্য। এখানে সংখ্যাও সংরক্ষণ করা যায়, যেমন:

```
>>> mylist = [ 1, 2, 5, 10, 20 ]
```

এবং স্ট্রিং হলো-

```
>>> mylist = [ 'a', 'bbb', 'ccccccc', 'dddddddddd' ]
```

এবং নাম্বার ও স্ট্রিং এর একটি মিশ্রণ হলো:

```
>>> mylist = [1, 2, 'a', 'bbb']
>>> print(mylist)
[1, 2, 'a', 'bbb']
```

এবং একটি লিস্টের লিস্ট হলো-

```
>>> list1 = [ 'a', 'b', 'c' ]
>>> list2 = [ 1, 2, 3 ]
>>> mylist = [ list1, list2 ]
>>> print(mylist)
[['a', 'b', 'c'], [1, 2, 3]]
```

উপরের উদাহরণে একটি ভেরিয়েবল 'list1 তৈরি করা হয়েছে তিনটি অক্ষর দিয়ে, 'list2' তৈরি করা হয়েছে তিনটি সংখ্যা দিয়ে এবং 'mylist' তৈরি করা হয়েছে list1 এবং list2 দিয়ে। আমরা যদি জিনিস দিয়ে ভেরিয়েবল তৈরি করি তবে তা অনেক সময় বিভ্রান্তিকর হয়, তাই যদি আমরা তালিকা তৈরি করি তবে তা যতগুলো তালিকাই হোক না কেন, পাইথন তা তৈরি করতে পারবে, এছাড়া এভাবে তালিকা তৈরি করে পাইথনে খুব সহজেই আমরা সকল আইটেম সহজেই সংরক্ষণ করতে পারি।

তালিকা শুধুমাত্র কেনাকাটার কাজেই প্রয়োজনীয় নয়।

আইটেমের প্রতিস্থাপন

পাইথনে খুব সহজে তালিকায় থাকা উপাদানের প্রতিস্থাপন করা যায়। আমরা যেভাবে ভেরিয়েবলের মান নির্ধারণ করি ঠিক সেই একই ভাবে তালিকায় থাকা উপাদানের মানের পরিবর্তন করা যায়। যেমন, আমরা তৃতীয় অবস্থানে থাকা banana কে পরিবর্তন করে chili করতে পারি:

```
>>> shopping_list[3] = 'chili'
```

```
>>> print(shopping_list)
[ 'eggs', 'milk', 'fish', 'chili', 'potato',  'meat' ]
```

একাধিক আইটেম যোগ করা...

পাইথনে করা তালিকায় আমরা 'append ' পদ্ধতি ব্যবহার করে একাধিক আইটেম যুক্ত করতে পারি। এই 'append' পদ্ধতি পাইথনকে বলে দেয় বা নির্দেশ দেয় তালিকায় আমরা কিছু করতে চাচ্ছি। আমরা এই পদ্ধতি সম্পর্কে পরে আরও জানবো কিন্তু এখন আমরা জানবো আমাদের করা বাজারের তালিকায় আমরা কীভাবে এই পদ্ধতি ব্যবহার করে একটি নতুন আইটেম যোগ করতে পারি, নিচের উদাহরণটি দেখ:

```
>>> shopping_list.append('mango')
>>> print(shopping_list)

[ 'eggs', 'milk', 'fish', 'chili', 'potato',  'meat', 'mango ]
```

লক্ষ কর, পূর্বের বাজার তালিকায় কোন পরিবর্তন না করে শুধু মাত্র তোমাদের পছন্দের একটি পণ্য 'আম' তালিকায় যুক্ত হয়েছে।

...এবং তালিকা থেকে কিছু মুছে ফেলা

আমরা 'del' (delete এর সংক্ষিপ্তরূপ) ব্যবহার করে খুব সহজেই তালিকায় থাকা কোন উপাদান মুছতে পারি। উদাহরণস্বরূপ তালিকার তৃতীয় স্থানে থাকা উপাদানটি (chili) মুছতে যা টাইপ করতে হবে:

```
>>> del shopping_list[3]
>>> print(shopping_list)
[ 'eggs', 'milk', 'fish', 'potato',  'meat', 'mango ]
```

আবারও তোমাদের মনে করিয়ে দিতে চাই তালিকা শুরু হয় শূন্য অবস্থান থেকে, তাই চতুর্থ স্থানে থাকা উপাদানটির তালিকায় অবস্থান হবে তৃতীয়।

একাধিক তালিকা একটির চেয়ে ভালো

তালিকায় থাকা আইটেমগুলো যোগ করে আমরা তালিকাগুলো যুক্ত করতে পারি, যেমন আমরা যদি দুইটি সংখ্যা যুক্ত করতে চাই তবে:

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 4, 5, 6 ]
>>> print(list1 + list2)
[1, 2, 3, 4, 5, 6]
```

আমরা দুইটি তালিকা যোগ করে তার ফলাফল অন্য একটি ভেরিয়েবলের সেট করতে পারি, যেমন:

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 4, 5, 6 ]
```

```
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 5, 6]
```

একই ভাবে আমরা তালিকার সাথে কোন স্ট্রিং গুণ করতে পারি:

```
>>> list1 = [ 1, 2 ]
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

উপরের উদাহরণে তালিকা এককে ৫ দিয়ে গুণ করার কথা বলা হয়েছে, অন্যভাবে বলা যেতে পারে, তালিকা এক ৫ বার পুনরাবৃত্তি করার কথা বলা হয়েছে। কিন্তু আমরা যখন তালিকা নিয়ে কাজ করবো তখন ভাগ (/) এবং বিয়োগ (-) এর কোন অর্থ আমরা পাবো না। তখন পাইথন কনসোল ত্রুটি দেখাবে, নিচের উদাহরণগুলো দেখ:

```
>>> list1 / 20
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

অথবা

```
>>> list1 - 20
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'type' and 'int'
```

এসব ক্ষেত্রে পাইথন আমাদের ত্রুটির বার্তা দেখাবে।

২.৭ টাপল এবং তালিকা

টাপল অনেকটা তালিকার মতোই, কিন্তু তা স্কয়ার বন্ধনীর [ ] সাহায্যে ব্যবহার করা হয়। আমরা যেমন তালিকাকে প্রথম বন্ধনী () দ্বারা প্রকাশ করি যেমন-' ('এবং')' তেমনি [ ] বন্ধনী দ্বারা টাপল প্রকাশ করা হয়। যেমন:

```
>>> t = (1, 2, 3)
>>> print(t[1])
2
```

এই দুইটির প্রধান পার্থক্য হলো তালিকার আইটেমগুলো পরিবর্তন করা যায় কিন্তু একবার টাপল তৈরি করা হলে তার আইটেমের মান আর পরিবর্তন করা যায় না। যদি পরিবর্তন করতে চাই তবে পাইথন কনসোল একটি বার্তা দেখাবে, যেমন:

```
>>> t[0] = 4
Traceback (most recent call last):
File "<stdin>", line 1, in ?
```

```
TypeError: 'tuple' object does not support item assignment
```

অবশ্য এর মানে এই নয় যে, টাপলের ভেতরের ভেরিয়েবল পরিবর্তন করা যাবে না, যেমন নিচের উদাহরণটি দেখ:

```
>>> myvar = (1, 2, 3)
>>> myvar = [ 'a', 'list', 'of', 'strings' ]
```

লক্ষ কর, প্রথমে আমরা একটি ভেরিয়েবল myvar তৈরি করেছি যা তিনটি টাপলকে নির্দেশ করে। এরপর আমরা myvar কে পরিবর্তন করেছি যা আরেকটি স্ট্রিং এর তালিকাকে নির্দেশ করে। প্রথম প্রথম এটি আমাদের বিভ্রান্তিতে ফেলে দেয়। কিন্তু তুমি যদি এভাবে চিন্তা কর, ধর তোমার বাসায় একটি লকার আছে। প্রতিটা লকারের উপরে একটি করে নাম লেখা আছে। তুমি লকারে কোন কিছু রেখে লকার বন্ধ করে তালা দিয়ে চাবি ফেলে দিলে। এরপর তুমি নাম ট্যাগটি খুলে ফেললে। এবার তুমি একটি খালি লকার খুঁজে পেলে যেখানে কোন কিছু নেই। আসলে এই দুটি লকারের মাঝে কোন পার্থক্যই নেই। একটিতে অনেক কিছু আছে কিন্তু তালা দেওয়া ও তোমার কাছে চাবি নেই। আরেকটি খালি লকার। সেরকম বলা যেতে পারে, টাপল হলো এমন লকার যা তালা দেওয়া। তুমি এর ভেতরের কিছু পরিবর্তন করতে পারছো না। কিন্তু তুমি তোমার লেভেলকে উঠিয়ে একটি তালা ছাড়া লকারে রাখতে পারো এবং এই লকারে তুমি তালিকার উপাদানগুলো রাখতে পারো।

২.৮ এসো নিজে করি

এই অধ্যায়ে আমরা দেখেছি কীভাবে পাইথন কনসোল ব্যবহার করে গাণিতিক সমীকরণের সমাধান করা যায়। আমরা আরো দেখেছি কীভাবে বন্ধনী ব্যবহার করে সমীকরণের ফলাফল পরিবর্তন করা যায়, কীভাবে বন্ধনী ব্যবহার করে অপারেটরের কাজকে নিয়ন্ত্রন করা যায়। এছাড়া আমরা দেখেছি পাইথন কীভাবে নির্ধারিত মানকে পরবর্তীতে ব্যবহার করা যায়, ভেরিয়েবলের ব্যবহার, কীভাবে অক্ষরকে সংক্ষিপ্ত করতে পাইথনে স্ট্রিং ব্যবহৃত হয়, কীভাবে তালিকা এবং টাপল ব্যবহার করে একাধিক তালিকা নিয়ে কাজ করা যায়। এবার আমরা নিজেরা কিছু করবো।

অনুশীলনী ১

তোমার প্রিয় খেলনার একটি তালিকা তৈরি করে এর নাম দাও toys। এবার তোমার প্রিয় খাবারসমূহের একটি তালিকা তৈরি করে এর নাম দাও foods। এবার দুইটি তালিকা সংযুক্ত কর এবং তার নাম দাও favourites, এবং এই নতুন তালিকাটি ফলাফল (print) এ দেখাও।

অনুশীলনী ২

ধর তোমার যদি তিনটি বাক্সে ২৫ টি চকোলেট এবং দশটি ব্যাগে ৩২ টি মিষ্টি থাকে তাহলে তোমার মোট কতগুলো মিষ্টি ও চকোলেট থাকবে।

(বি.দ্র.: তুমি এটি পাইথন কনসোল ব্যবহার করে একটি সমীকরণেই করতে পারবে।)

অনুশীলনী ৩

তোমার নামের প্রথম ও শেষ অংশ ব্যবহার করে একটি ভেরিয়েবল তৈরি কর। এখন একটি স্ট্রিং তৈরি কর এবং এতে তোমার নাম যুক্ত করতে একটি প্লেস হোল্ডার ব্যবহার কর।

তৃতীয় অধ্যায়

কাছিম এবং অন্যান্য ধীর গতির প্রাণী

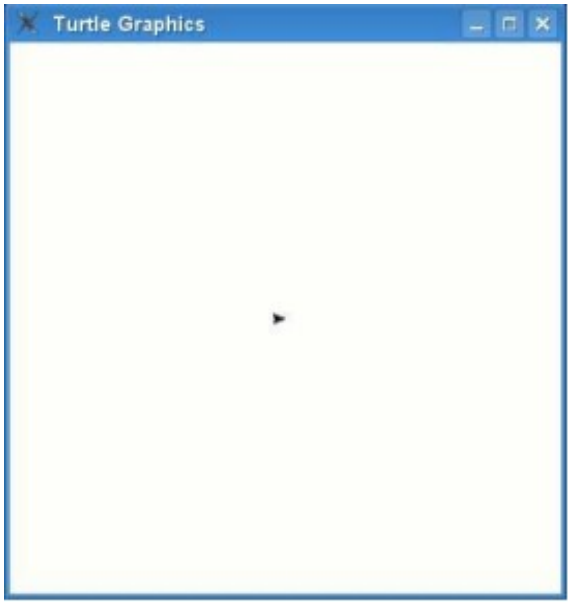
আমি ছোটবেলায় একটা কার্টুন ছবি দেখতাম, নিনজা টার্টলস্। তোমরা কি দেখেছ? তোমরা নিশ্চই বাস্তবে কাছিম দেখে থাকবে। পাইথনের কাছিমের সাথে বাস্তবের কাছিমের অনেক মিল রয়েছে। বাস্তবে কাছিম কেমন হয়? নিনজা টার্টলস্ এর মত সবুজ? হ্যাঁ, তা ধরে নিতে পার। কাছিম আসলে এক প্রকার সরীসৃপ যা অত্যন্ত ধীর গতিতে চলাফেরা করে এবং পিঠে করে যাযাবরের মত তার বাসস্থান বহন করে। আর পাইথনের দুনিয়ায়, কাছিম হল এক প্রকার কালো রঙের তীর যা কিনা কম্পিউটারের পর্দায় খুব ধীর গতিতে নড়াচড়া করে। যদিও এখানে কোন ঘরবাড়ি বহন করার ব্যপার নেই।

আসলে, ধরে নিতে পার পাইথনের কাছিম কম্পিউটারের পর্দায় নড়াচড়া করার সময় একটি চিহ্ন রেখে যায়, এই ব্যাপারটার জন্য বাস্তবের কাছিমের সাথে এর অমিল চোখে পড়ে। হয়তো অন্য কোন প্রাণী আছে যারা তাদের চলাফেরার সময় চিহ্ন রেখে যায়। তবে সেগুলো হয়তো আমাদের পরিচিত নয়। তোমরা কি তেমন কোন প্রাণীর নাম মনে করতে পারছো? যদি পারো তাহলে আমাকে বলতে পারো। আর যেহেতু আমরা এখনো জানি না তাই আমরা কাছিম (turtles) নিয়েই থাকব। এখন কল্পনা কর, একটি কাছিম তার সাথে দুটো মার্কার কলম বহন করছে এবং সে সামনের দিকে আঁকতে আঁকতে যাচ্ছে।

অনেক অনেক কাল আগে, লোগো নামে একটি সাধারণ প্রোগ্রামিং ভাষা ছিল। আরভিং নামক একটি রোবোট কাছিম ছিল। একে কন্ট্রোল করার জন্য লোগো ব্যবহৃত হত। বর্তমানে আমাদের কম্পিউটারে যেভাবে মাউস নড়াচড়া করতে পারে, সময়ের সাথে সাথে আরভিংও সফল হয়েছিলো মেঝেতে সেভাবে নড়াচড়া করতে।

এখান থেকে এটাই প্রমাণ হয়, প্রযুক্তির বিকাশের সাথে সাথে সবসময় চাহিদার পূরণ হয় না --- একটি ছোট্ট রোবট কাছিম অনেক মজার ব্যাপার হতে পারত।

পাইথনের টার্টল (turtle) একটি মডিউল (মডিউল কি সেটা নিয়ে আমরা পরে আলোচনা করব, আপাতত ধরে নাও



চিত্র: ৩.১: একটি তীর যা টার্টল এর প্রতিরূপ

মডিউল হচ্ছে কম্পিউটারের প্রোগ্রামে ব্যবহৃত হয় এমন এমন একটি জিনিস)। যা কিছুটা লোগো প্রোগ্রামিং ভাষার মত, কিন্তু লোগো'র ছিল সীমাবদ্ধতা, আর পাইথনের রয়েছে অপরিসীম ক্ষমতা। টার্টল মডিউল ব্যবহারের মাধ্যমে তোমরা শিখতে পারবে কম্পিউটারের পর্দায় কীভাবে ছবি আঁকাতে হয়।

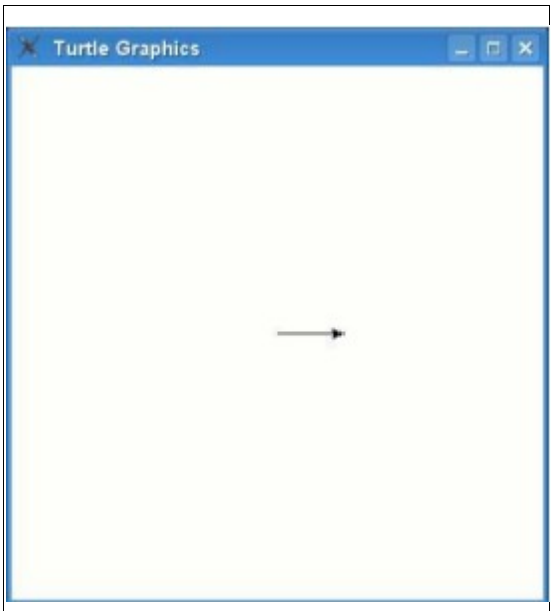
চল তাহলে শুরু করা যাক। আমরা দেখব কীভাবে এটি কাজ করে। প্রথম কাজ হচ্ছে পাইথনকে বলা আমরা টার্টল ব্যবহার করতে চাই, এর জন্য মডিউল ইমপোর্ট করতে হবে:

```
>>> import turtle
```

এবার আমাদের একটি ক্যানভাস দরকার ছবিটি আঁকার জন্য। ক্যানভাস নিশ্চয়ই তোমরা চেন, আঁকিয়েরা ছবি আঁকার জন্য যেটি ব্যবহার করে থাকে:

```
>>> t = turtle.Pen()
```

এই কোডে, আমরা টার্টল মডিউলে একটি বিশেষ ফাংশন (Pen) কে ডেকেছি, যা স্বয়ংক্রিয়ভাবে একটি ক্যানভাস তৈরি করবে যার উপর আমরা ছবি আঁকব। ফাংশন হচ্ছে বারবার ব্যবহারযোগ্য এমন এক ধরনের কোড (আমরা ফাংশন নিয়ে পরবর্তীতে বিশদ আলোচনা করব) যা অনেক কার্যকরী --- এ ক্ষেত্রে, টার্টেলের অবজেক্ট Pen ফাংশনের মাধ্যমে ফেরত এসেছে --- আমরা অবজেক্ট হিসেবে নির্ধারণ করেছিলাম 't' নামক ভ্যারিয়েবল (ফলাফলে, আমাদের পুরো ক্যানভাসই 't' নামে পরিচিত হচ্ছে)। যখন তোমরা কোডটি পাইথন কনসোলে লিখবে, তখন চিত্র ৩.১ এর মত একটি খালি বক্স (ক্যানভাস) দেখতে পাবে।।



চিত্র ৩.২: কাছিমের আঁকা একটি রেখা

হ্যাঁ, পর্দার মাঝে এই ছোট তীরটিই হল সত্যি সত্যি একটি কাছিম কিন্তু না, এটি দেখতে মোটেও কাছিমের মত নয়।

তোমরা এখন কাছিম টিকে নির্দেশ দিতে পার, সেই ফাংশনটি ব্যবহার করে যেটি তোমরা একটু আগেই তৈরি করলে (turtle.Pen)---যেহেতু আমরা অবজেক্ট হিসেবে ভ্যারিয়েবল 't' নির্বাচন করেছিলাম, আমরা নির্দেশ দেয়ার জন্য এই 't' ব্যবহার করব। চল তাহলে একটা নির্দেশ দিই, forward। 'ফরওয়ার্ড' নির্দেশটি কাছিম যে অবস্থানেই থাকুক না কেন, তাকে সামনে সরে যেতে বলবে (আমার কোন ধারণা নেই কাছিমটি মেয়ে নাকি ছেলে, কিন্তু আপাতত ধরে নেয়া যাক এটি একটি মেয়ে কাছিম)।

চল আমরা কাছিম কে বলি সামনের দিকে ৫০ পিক্সেল সরে যেতে (একটু পরেই আমরা পিক্সেল নিয়ে আলোচনা করব):

```
>>> t.forward(50)
```

এই কোডটি লিখলে তোমরা চিত্র ৩.২ এর মত কিছু একটা দেখতে পাবে।

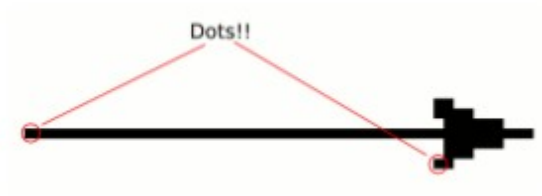
যদি আমরা কাছিমের ভাষায় বলতে চাই, তাহলে বলব, কাছিমটি সামনের দিকে ৫০ টি পদক্ষেপ এগিয়েছে। আর

যদি আমরা আমাদের মতো করে বলি, তাহলে বলব, কাছিমটি ৫০ পিক্সেল সরেছে।

তাহলে, পিক্সেল কী?

একটি পিক্সেল হচ্ছে পর্দার উপর একটি ডট। কম্পিউটারের পর্দার দিকে তাকিয়ে তোমারা যা দেখবে তার সবকিছুই ছোট ছোট ডটের (বর্গাকৃতির) সমষ্টি। তোমরা কম্পিউটারে অথবা প্লেস্টেশনে (Playstation) অথবা এক্সবক্সে (Xbox) অথবা উই-তে (Wii) যে প্রোগ্রাম ব্যবহার কর এবং যে গেইম খেল তার সবকিছুই তৈরি হয়েছে বিভিন্ন রঙের অনেক অনেক ডটের সমষ্টি দ্বারা। তোমরা ম্যাগনিফাইং গ্লাস ব্যবহার করে যদি কম্পিউটারের পর্দা দেখ তাহলে অসংখ্য ডটের সমষ্টি দেখতে সক্ষম হবে। এখন আমরা যদি আমাদের ক্যানভাসটিকে বড় করে দেখি তাহলে দেখব সেই তীরটি অর্থাৎ কাছিমটি এবং সে যে লাইনটি এঁকেছিল তা আসলে কয়েকটি ডটের সমষ্টি, যেরকমটা আমরা চিত্র ৩.৩ এ দেখতে পাচ্ছি।

আমরা পিক্সেল এবং ডট নিয়ে পরবর্তী অধ্যায়গুলোতে আলোচনা করব। এখন আমরা কাছিমকে বলব বাম অথবা ডান দিকে যেতে:



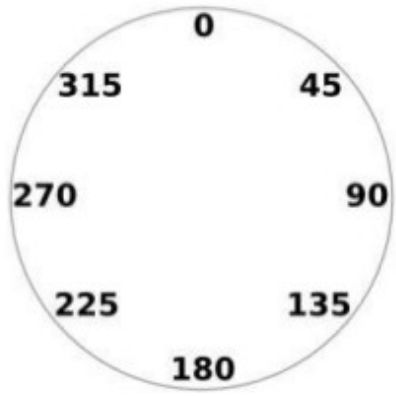
চিত্র ৩.৩: তীর এবং রেখাটি বড় করে দেখানো হয়েছে

চিত্র ৩.৪: একটি ঘড়ির ভাগসমূহ (divisions)

```
>>>  t.left(90)
```

এই কোড কাছিম কে বলবে "তুমি তোমার অবস্থান থেকে বাম দিকে ৯০ ডিগ্রী ঘুরে যাও"। তুমি হয়ত স্কুলে এখনও ডিগ্রী সম্পর্কে পড় নি, কিন্তু সবচেয়ে সহজভাবে এ সম্পর্কে চিন্তা করতে পারবে, যদি তোমরা একটি ঘড়ির দিকে তাকাও এবং এর ভাগগুলো দেখ, ঠিক চিত্র ৩.৪ এর মত।

ঘড়ির সাথে এর পার্থক্য হচ্ছে, ১২ টি ভাগের পরিবর্তে (অথবা ৬০ টি ভাগ, যদি তোমরা মিনিটে গণনা কর), এতে রয়েছে ৩৬০ টি ভাগ। তাই তোমরা যদি ঘড়ির দিকে তাকিয়ে ৩৬০ টি ভাগ গণনা কর, তোমরা পাবে ৯০ যেখানে সাধারণত থাকে ৩, যদি ১৮০ টি ভাগ গণনা কর, তাহলে পাবে ঘড়ির ৬, এবং ২৭০ গণনা করলে পাবে ৯; এবং ০



চিত্র: ৩.৫: ডিগ্রী

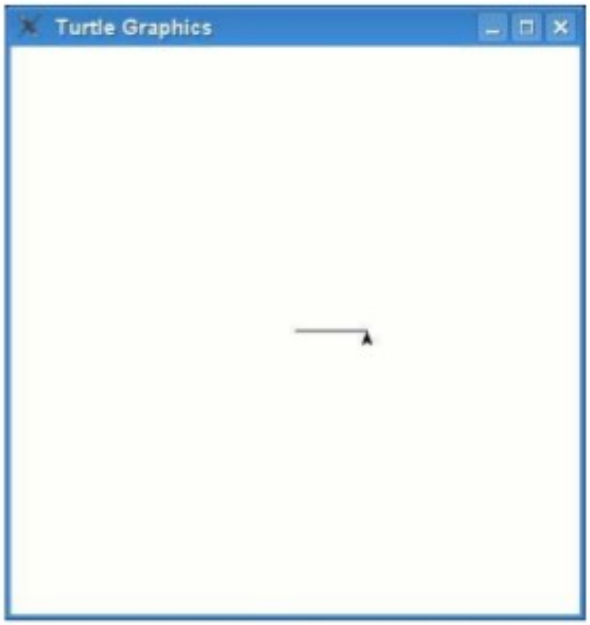
পাবে একদম উপরে, যেখানে তোমরা সাধারণত দেখ ১২। চিত্র ৩.৫ এ তোমরা ডিগ্রীর ভাগগুলো দেখতে পাবে। সুতরাং, t.left(90) এর ফলাফল হচ্ছে তীরটি উপরের দিকে মুখ করে থাকবে, চিত্র ৩.৬ এর মত।

এসো আমরা একই কমান্ড আরও কয়েকবার ব্যবহার করি:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

আমাদের কাছিমটি একটি বর্গক্ষেত্র অঙ্কন করেছে এবং বাম দিকে মুখ করে আছে অর্থাৎ যেভাবে সে শুরু করেছিল (চিত্র ৩.৭ দেখ)।

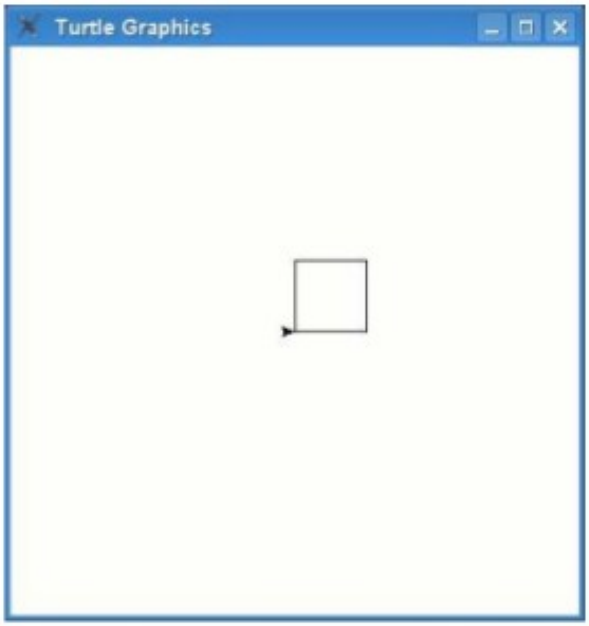
আমরা clear ব্যবহার করে ক্যানভাসের ছবি মুছে ফেলতে পারব:



৩.৬: বাম দিকে ঘোরার পর কাছিমের অবস্থান

```
>>> t.clear()
```

অন্যান্য আরও বেসিক ফাংশন এই কাছিমের সাহায্যে তোমরা ব্যবহার করতে পার: reset, এটাও পুরো ক্যানভাসের সব ছবি মুছে ফেলবে কিন্তু কাছিমটিকে প্রারম্ভিক স্থানে নিয়ে যাবে; backward, যা কাছিমটিকে পেছনের দিকে নড়াচড়া করতে সাহায্য করে; right, যা কাছিমকে ডান দিকে ঘুরায়; up, যা কাছিমকে নড়াচড়া করার সাথে সাথে ছবি আঁকা বন্ধ করতে নির্দেশ দেয় (অন্যভাবে বললে, কাছিমের কলমটি তুলে নেয়া); down, যা কাছিমকে পুনরায় ছবি আঁকার নির্দেশ দেয়। এখানে যেভাবে অন্য ফাংশন এর ব্যবহার দেখানো হয়েছে তোমরা



চিত্র ৩.৭: বর্গক্ষেত্র অঙ্কন

একইভাবে এই ফাংশনগুলোও ব্যবহার করতে পারবে:

৩.১। তোমরা এবার নিজেরা করার চেষ্টা কর:

```
>>> t.reset()
>>> t.backward(100)
>>> t.right(90)
>>> t.up()
>>> t.down()
```

আমরা একটু পরে আবারও turtle মডিউলে ফিরে আসব।

৩.১ তোমরা চেষ্টা কর

আমরা এই অধ্যায়ে দেখেছি কীভাবে একটি কাছিম দ্বারা লাইন আঁকতে হয়, বাম এবং ডান দিকে ঘুরানোর মাধ্যমে। আমরা দেখেছি কাছিম ঘুরার জন্য ডিগ্রী ব্যবহার করে, অনেকটা ঘড়ির মধ্যে মিনিটের ভাগগুলোর মত।

অনুশীলনী ১

turtle এর Pen ফাংশন ব্যবহার করে একটি ক্যানভাস তৈরি কর এবং একটি আয়তক্ষেত্র আঁক।

অনুশীলনী ২

turle এর Pen ফাংশন ব্যবহার করে আরেকটি ক্যানভাস তৈরি কর এবং একটি ত্রিভূজ আঁক।

চতুর্থ অধ্যায়

পাইথনে প্রশ্ন করার পদ্ধতি

আমরা অনেক সময়ই অনেক কিছু জানার জন্য প্রশ্ন করি। আবার কেউ কেউ কখনো কখনো আমাদের প্রশ্ন করে। কিন্তু আমরা কি কখনো প্রশ্ন করার আগে উত্তর কী হবে তা নিয়ে ভাবি? বুঝতে একটু সমস্যা হচ্ছে? এসো আবার আমরা একটু চিন্তা করি। আমরা যখন কোন কছু জানতে চাই তখন প্রশ্ন করি। কিন্তু উওর ঠিক করে তারপর আমরা প্রশ্ন ঠিক করি না কখনোই। বা প্রশ্ন করার আগে উত্তর নিয়ে আমরা কখনোই ভাবি না। প্রোগ্রামিং এ প্রশ্ন করা মানে হল, উত্তর দেয়ার ধরনটা আগে ভাগে চিন্তা করে প্রশ্ন করা। যেমন:

তোমার বয়স কত? যদি তুমি ২০ বছরের বেশি হও, তাহলে তুমি বুড়ো!

এটি যদি আমরা পাইথনে লিখি তাহলে আমাদের যা লিখতে হবে, তা হলো:

```
if age > 20:
    print('you are too old!')
```

এখানে যা ব্যবহার করা হয়েছে তা হল একটি if-statement। একটি if-statement শুরু হয় if দিয়ে এবং তার পরপরই থাকে শর্ত অর্থাৎ 'condition' (এই ব্যপারে একটু পরেই বিস্তারিত জানব), এবং তার পর থাকে কোলন ':'। পরবর্তী লাইন হতে হবে অবশ্যই একটি ব্লক এর মধ্যে এবং যদি প্রশ্নটির উত্তর হয় 'হ্যাঁ' অথবা 'সত্য' (যেভাবে আমরা প্রোগ্রামিং-এ সচরাচর দেখে থাকি) তাহলে ব্লকের মধ্যের কমান্ডটি সক্রিয় হবে। অর্থাৎ প্রথমে আমরা if-statement এ শর্ত দিচ্ছি এবং শর্ত যদি মিলে যায় তাহলে যে কাজটি আমরা করাতে চাই সেটা ব্লকের ভেতর রাখছি।

Condition হল একটি প্রোগ্রামিং statement যা উত্তর হিসেবে আমাদেরকে 'yes' (True) অথবা 'no' (False) রিটার্ন (return) করে। শর্ত আরোপ করার জন্য আমরা কিছু প্রতীক/symbol ব্যবহার করি। এখন তোমাকে জানতে হবে শর্ত আরোপ করার জন্য ব্যবহৃত symbol (অথবা operators) গুলো কি কি:

==	সমান/equals
!=	সমান নয়/Not equals
>	'...' এর চেয়ে বড়/Greater than
<	'...' এর চেয়ে ছোট/Less than
>=	'...' এর চেয়ে বড় বা সমান/Greater than or Equal to
<=	'...' এর চেয়ে ছোট বা সমান/Less than or Equal to

উদাহরণস্বরূপ, যদি তোমার বয়স হয় ১০, তাহলে তাহলে তোমার বয়সের condition your_age==10 রিটার্ন করবে True (yes), কিন্তু যদি তুমি ১০ না হও, এটি রিটার্ন করবে False। মনে রাখবে: প্রোগ্রামে দুটো সমান চিহ্ন যুক্ত (==) শর্ত অর্থাৎ condition দেয়ার সময় একটি সমান চিহ্ন (=) দেওয়া যাবে না, যদি দাও তাহলে প্রোগ্রামে ত্রুটি (error) দেখাবে।

ধর তোমার বয়স ১২ এবং age নামক একটি variable এ তোমার বয়স রাখা হয়েছে বা প্রোগ্রামিং-এর ভাষায় আমরা বলতে পারি set করা হয়েছে এবং condition দেওয়া হয়েছে এরকম...

```
age > 10
```

....উত্তর পাবে 'True'। যদি তোমার বয়স ৮ হয়, তাহলে এটি উত্তর দিবে 'False'। যদি তোমার বয়স ১০ হয় তাহলেও উত্তর পাবে 'False'। কারণ condition টি পরীক্ষা করছে তোমার বয়স ১০ এর থেকে বড় কিনা। আমরা আরও কিছু উদাহরণ দেখি:

```
>>> age=10
>>> if age>10:
...  print('got here')
```

যদি তুমি প্রোগ্রামটা রান (run) কর তাহলে দেখবে 'got here' মেসেজটি কনসোলে প্রিন্ট হয়েছে। একই জিনিস হবে যদি তোমরা পরবর্তী উদাহরণটা রান করো:

```
>>> age=10
>>> if age==10:
...  print('got here')
```

৪.১ এটি অথবা অন্যটি কর (Do this... or ELSE) !!!

আমরা if-statement কে আরও প্রসারিত করতে পারি, যাতে করে শর্ত সত্যি না হলে এটি অন্য কিছু করে। উদাহরণস্বরূপ, যদি তোমার বয়স ১২ হয় তাহলে 'Hello' প্রদর্শিত হবে, কিন্তু যদি তা না হয় তাহলে প্রদর্শিত হবে 'Goodbye'। এ কাজ টি করার জন্য আমরা ব্যবহার করব if-then-else-statement (অন্যভাবে বললে বিষয়টি দাঁড়ায়: " যদি কোন কিছু সত্য হয়, তাহলে একটি কাজ কর, আর যদি তা না হয়, তাহলে অন্য একটি কাজ কর"):

```
>>> age = 12
>>> if age == 12:
...  print('Hello')
...  else:
...  print('Goodbye')
Hello
```

উপরের কোডটি যদি তোমরা রান করো তাহলে দেখবে 'Hello' প্রদর্শিত হচ্ছে। তোমরা যদি বয়স ১২ এর জায়গায় অন্য কিছু দাও তাহলে 'Goodbye' প্রদর্শিত হবে:

```
>>> age = 8
>>> if age == 12:
...  print('Hello')
...  else:
...  print('Goodbye')
Goodbye
```

৪.২ এটি করো.. অথবা ওটি করো... অথবা সেটি করো... অথবা অন্যটি করো!!! (Do this... or do this... or do this... or ELSE!!!)

আমরা চাইলে এই if-statement টিকে আরও বড় করতে পারি elif ব্যবহার করে (elif হচ্ছে else-if এর সংক্ষিপ্ত রূপ)। উদাহরণস্বরূপ, আমরা পরীক্ষা করতে পারি তোমার বয়স কি ১০?, অথবা ১১?, অথবা ১২? ইত্যাদি:

```
1. >>> age = 12
2. >>> if age == 10:
3. ...    print('You are 10')
4. ...   elif age == 11:
5. ...    print('you are 11')
6. ...  elif age ==12:
7. ...    print('You are 12')
8. ...  elif age == 13:
9. ...    print('You are 13')
10. ...  else:
11. ...   print('huh?')
12. ...
You are 12
```

উপরের কোডে ২ নম্বর লাইনটি পরীক্ষা করছে যে তোমার বয়স অর্থাৎ age নামক variable এর মান ১০ এর সমান কিনা। কিন্তু তুমি তোমার বয়স দিয়েছিলে ১২, তাই পাইথন সরাসরি ৪ নম্বর লাইনে গিয়ে আবার পরীক্ষা করবে age এর মান কত এর সমান। এখানেও মিলে নি তাই আবার লাফ দিয়ে চলে যাবে ৬ নম্বর লাইনে, হ্যাঁ এবারে মিলেছে। এখন প্রদর্শিত হবে 'You are 12'। (তোমরা নিশ্চয়ই লক্ষ করেছ এই কোডে ৫ টি গ্রুপ রয়েছে: লাইন ৩,৫,৭,৯ এবং ১১)

৪.৩ দুটো শর্ত একসাথে

তোমরা দুটো অথবা তারচেয়েও বেশী শর্ত একসাথে জুড়ে দিতে পার 'and' এবং 'or' ব্যবহার করে। এসো আমরা আগের উদাহরণটিকে শর্ত জোড়া দিয়ে ছোট করে ফেলি:

```
1. >>> if age == 10 or age == 11 or age == 12 or age == 13:
2. ...    print('You are %s' % age)
3. ... else:
4. ...    print('huh?')
```

যদি ১ নম্বরলাইনের যেকোন একটি শর্ত সত্যি হয় (অর্থাৎ if age is 10 or age is 11 or age is 12 or age is 13), তাহলে ২ নম্বর লাইনের ব্লকের কোড টি রান হবে, অন্যথায় পাইথন সরাসরি চলে যাবে ৪ নম্বর লাইনে।। আমরা উদাহরণটি আরও ছোট করে ফেলতে পারি 'and', >= এবং <= চিহ্ন ব্যবহারের মাধ্যমে:

```
1. >>> if age >= 10 and age <=13:
2. ...    print('You are %s' %age)
3. ... else:
4. ...    print('huh?')
```

আশা করছি তোমরা বুঝতে পেরেছ, যদি এক নম্বর লাইনের দুটো শর্তই সত্যি হয় তাহলে ২ নম্বর লাইনের ব্লকের কোড রান হবে। কারণ ১২ অবশ্যই ১০ এর থেকে বড় এবং ১৩ এর থেকে ছোট।

৪.৪ একাকীত্ব

আরেক ধরনের মান আছে, যা ভ্যারিয়েবলে নির্দিষ্ট করে দেয়া যায়, যা আমরা পূর্ববর্তী অধ্যায়ে আলোচনা করিনি। তা হল: Nothing.
যেভাবে আমরা ভ্যারিয়েবলের মান নির্ধারণ করে দেই সেভাবেই আমরা ভ্যারিয়েবলে 'nothing' নির্ধারণ করে দিতে পারি। পাইথনে কোন মান ছাড়া অর্থাৎ empty রাখলে তা None হিসেবে নির্ধারিত হয় (অন্যান্য প্রোগ্রামিং ভাষায় এটাকে মাঝে মধ্যে বলা হয় null) এবং তুমি এটাকে অন্যান্য মানের মত বসাতে পারবে:

```
>>> myval = None
>>> print(myval)
None
```

None ব্যবহার করে একটি ভ্যারিয়েবল যার পূর্বে কোন মান ছিল তা পুনরায় নির্ধারণ/reset করা যায়, অথবা কোন ভ্যারিয়েবল ব্যবহার হওয়ার পূর্বে তার মান নির্ধারণ না করে None হিসেবে ব্যবহার করা যায়।

উদাহরণস্বরূপ, তোমাদের ফুটবল খেলার দল তাদের নতুন জার্সির জন্য যখন ফান্ড সংগ্রহ করতে থাকে তখন তোমাকে বলা হল হিসাব রাখার জন্য, তুমি তো জান না কোন খেলোয়ার কত টাকা করে দিয়েছে তাই তুমি প্রত্যেক খেলোয়ারের জন্য একটি করে ভ্যারিয়েবল নির্ধারণ করে তাদের মান হিসেবে None ব্যবহার করতে পার। পরবর্তীতে যখন ফান্ড সংগ্রহ হতে থাকবে তখন তুমি None এর জায়গায় প্রাপ্ত মান বসিয়ে নিতে পারবে:

```
>>> player1 = None
>>> player2 = None
>>> player3 = None
```

তারপর আমরা সেই আগের মত if-statement ব্যবহার করে পরীক্ষা করতে পারব ফান্ড সংগ্রহ হয়েছে কিনা অথবা কতটুকু হয়েছে তার হিসাব:

```
>>> if player1 is None or player2 is None or player3 is None:
...  print('Please wait until all players have returned')
... else:
...  print('You have raised %s'%(player1+player2+player3))
```

if-statement টি পরীক্ষা করে ভ্যারিয়েবলে কোন মান রয়েছে নাকি None হিসেবে সেট করা হয়েছে, যদি none থাকে তাহলে প্রথম ম্যাসেজটি আউটপুট দিবে। যদি সবগুলো ভ্যারিয়েবলের জন্য নির্ধারিত মান থাকে তাহলে দ্বিতীয় ম্যাসেজটি আউটপুট দিবে।

অর্থাৎ যদি সব ভ্যারিয়েবলের মান None হিসেবে নির্ধারণ করা হয় তাহলে যা হবে:

```
>>> if player1 is None or player2 is None or player3 is None:
...  print('Please wait until all players have returned')
... else:
...  print('You have raised %s'%(player1+player2+player3))
Please wait until all players have returned
```

কোডটি লিখার সময় প্রথমে ভ্যারিয়েবল তৈরি করতে ভুলে যেও না, কারণ তখন Error ম্যাসেজ দেখতে পাবে।

যদি আমরা দুটো ভ্যারিয়েবলেরও নির্দিষ্ট মান দেই তাহলেও আমরা একই ম্যাসেজ দেখতে পাব:

```
>>> player1 = 100
>>> player3 = 300
>>> if player1 is None or player2 is None or player3 is None:
...  print('Please wait until all players have returned')
...  else:
...  print('You have raised %s' % (player1 + player2 +
player3))
Please wait until all players have returned
```

অবশেষে, যখন সবগুলো ভ্যারিয়েবলের মান নির্ধারিত হবে তখন আমরা দ্বিতীয় ব্লকের ম্যাসেজ দেখতে পাব:

```
>>> player1 = 100
>>> player3 = 300
>>> player2= 500
>>> if player1 is None or player2 is None or player3 is None:
...  print('Please wait until all players have returned')
...  else:
...  print('You have raised %s' % (player1 + player2 +
player3))
You have raised 900
```

৪.৫ পার্থক্য কী ...?

10 এবং '10' এর মধ্যে পার্থক্য কী?

তোমরা হয়ত ভাববে কোটেশন '... ...' ছাড়া আর কোন পার্থক্য নেই। ভাল কথা, পূর্বের অধ্যায় যদি পড়ে থাক তাহলে তোমরা এটাও জানবে যে, প্রথমটি একটি সংখ্যা এবং দ্বিতীয়টি একটি স্ট্রিং। তুমি যতটুকু ভাবছো এক্ষেত্রে তার থেকেও বেশী পার্থক্য রয়েছে। আগে আমরা একটি ভ্যারিয়েবলের মানের (age) সাথে একটি সংখ্যার তুলনা করেছিলাম if-statement এর মধ্যে:

```
>>> if age == 10:
...  print('you are 10')
```

যদি তোমরা ভ্যারিয়েবল age এর মান 10 নির্ধারণ করে দাও, তাহলে যা আউটপুটে আসবে:

```
>>> age = 10
>>> if age == 10:
...  print('you are 10')
...
you are 10
```

কিন্তু যদি age নির্ধারণ কর '10' তাহলে আউটপুটে এটি দেখাবে না:

```
>>> age = '10 '
>>> if age == 10:
...  print(’you are 10’)
...
```

কেন আউটপুট হল না? কারণ একটি স্ট্রিং এবং একটি সংখ্যার মধ্যে পার্থক্যের কারণে যদিও তারা দেখতে একইরকম:

```
>>> age1 = 10
>>> age2 = ’10’
>>> print(age1)
10
>>> print(age2)
10
```

দেখেছ? তারা দেখতে পুরোপুরি একইরকম। তারপরেও যেহেতু একটি হচ্ছে সংখ্যা এবং অন্যটি স্ট্রিং তাই তারা আলাদা মান। সুতরাং age==10 (age equals 10) কখনও সত্যি হবে না, যদি ভ্যারিয়েবলের মান একটি স্ট্রিং হয়।

ধরে নাও তোমার কাছে ১০ টি বই আছে এবং ১০ টি ইট আছে। দুটো জিনিসের সংখ্যার পরিমান সমান, কিন্তু তুমি বলতে পারবে না ১০ টি বই এবং ১০ টি ইট এক সমান, পারবে কি? সৌভাগ্যবশত আমাদের পাইথনে এমন একধরনের যাদু আছে যা স্ট্রিং-কে সংখ্যায় এবং সংখ্যাকে স্ট্রিং-এ পরিণত করতে পারে (আবার ভেবো না যে, বই কে ইট আর ইট কে বই বানিয়ে দেয়া হবে)। উদাহরণস্বরূপ, স্ট্রিং '10' কে সংখ্যায় পরিবর্তন করতে আমরা একটি ফাংশন ব্যবহার করব:

```
>>> age = '10'
>>> converted_age = int(age)
```

ভ্যারিয়েবল converted_age এখন 10 ধরে আছে, এবং এটি স্ট্রিং নয়। যদি আমরা একটি সংখ্যাকে স্ট্রিং-এ রূপান্তরিত করতে চাই তাহলে আমরা ব্যবহার করব str ফাংশন:

```
>>> age = 10
converted_age = str(age)
```

ভ্যারিয়েবল converted_age এখন '10' ধরে আছে, এবং এটি সংখ্যা নয়। এবার আমরা সেই if-statement এ যাই যেটা কিছুই আউটপুট দেয়নি:

```
>>> age = '10'
>>> if age == 10:
...  print('you are 10')
...
```

যদি পরীক্ষা করার আগেই আমরা ভ্যারিয়েবল পরিবর্তন করি তাহলে আমরা ভিন্ন ফলাফল পাব:

```
>>> age = '10'
>>> converted_age = int(age)
>>> if converted_age == 10:
...  print('you are 10')
...
you are 10
```

পঞ্চম অধ্যায়

আবার এবং আবার

একই জিনিস বারবার করার চেয়ে আর খারাপ কিছু হতে পারে না। একটু মনে করে দেখো, যখন তোমার মা-বাবা তোমাকে ঘুম আসার জন্য ভেড়া গুণতে বলেন বা সংখ্যা গুণতে বলেন তখন কী হয়? তখন তোমার ঘুম চলে আসে, তাই না? ভেবে দেখো এই ভেড়া বা পশমী স্তন্যপায়ী প্রাণীর পক্ষে আশ্চর্যজনক ঘুম নিয়ে আসা কি আসলেই সম্ভব? ভেড়ার সাথে নিশ্চয় ঘুমের কোন সম্পর্ক নেই। মূল কথা হলো কোন কিছু বারবার করা বিরক্তিকর আর এই বিরক্তিকর কাজটি যখন আমরা করি তখনই আমাদের ঘুম পায় এবং এজন্যই সহজে আমরা ঘুমিয়ে পরি, তবে যদি এই কাজটি অর্থাৎ ভেড়া গোনার কাজটি করাতে কারো ভালো লাগে বা আকর্ষণীয় মনে হয় তাহলে কিন্তু তার আর ঘুম পাবে না।

প্রোগ্রামাররা কোড লেখার সময় একই কাজ বারবার করাটা পছন্দ করেন না। একই কাজ বারবার করা পছন্দ না করাটা আবার তাদের অলস করতেও সাহায্য করে। প্রায় সকল প্রোগ্রামিং ভাষার কোডিং-এ একই কাজের পুনরাবৃত্তি না ঘটার জন্য একটি বিষয় ব্যবহার করা হয় যাকে বলে for-loop। উদাহরণস্বরূপ, তোমাকে পাইথনে hello পাঁচবার প্রিন্ট করার কোড লিখতে বলা হলে, তুমি হয়তো নিচের কোডটি ব্যবহার করবে-

```
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
>>> print("hello")
hello
```

একে বিরক্তিকর বা একঘেয়েমি বলা যেতে পারে...

এই কাজটি না করে আমরা for-loop (ফর-লুপ) ব্যবহার করতে পারি। (নোট: এখানে দ্বিতীয় লাইনের print স্টেটমেন্টের পূর্বে আমরা চারটি spaces বা ফাঁকা স্থান ব্যবহার করা হয়েছে, আমরা এখানে @ সিম্বলটি ব্যবহার করেছি যাতে করে সহজেই তোমরা বুঝতে পারো স্থানটি কোথায়):

```
>>> for x in range(0, 5):
... @@@@print('hello')
...
hello
hello
hello
hello
hello
```

উপরের উদারহণটিতে লক্ষ করো range শব্দটি ব্যবহৃত হয়েছে। range হলো একটি ফাংশন যেটি খুব সহজেই প্রথম ও শেষ সংখ্যার একটি তালিকা তৈরি করতে পারে। উদাহরণস্বরূপ:

```
>>> print(list(range(10, 20)))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

সুতরাং for-loop এর ক্ষেত্রে বলা যায় "for x in range(0,5)" কোডটি পাইথনকে নির্দেশ করে সংখ্যার একটি তালিকা (0, 1, 2, 3, 4) তৈরি করো এবং ভ্যারিয়েবল X এর মধ্যে প্রতিটি সংখ্যার মান সংরক্ষণ করো। এরপর যদি আমরা চাই X কে আমরা print স্টেটমেন্টে ব্যবহার করতে পারি:

```
>>> for x in range(0, 5):
... print('hello %s' % x)
hello 0
hello 1
hello 2
hello 3
hello 4
```

যদি আমরা for-loop থেকে অব্যহতি পেতে চাই কিংবা for-loop ব্যবহার করতে না চাই তাহলে উপরের কোডটি হবে এরূপ:

```
x = 0
print('hello %s' % x)
x = 1
print('hello %s' % x)
x = 2
print('hello %s' % x)
x = 3
print('hello %s' % x)
x = 4
print('hello %s' % x)
```

সুতরাং তোমরা একটি বিষয় লক্ষ্য করলে তো, loop আমাদেরকে বেশকিছু লাইনের একইরকম কোড লেখা হতে বাঁচিয়ে দিল। এটি খুবই প্রয়োজনীয়, কেননা বেশির ভাগ প্রোগ্রামাররা কোড লেখার কাজে জলহস্তীর চেয়ে বেশি অলস হয়। ভাল প্রোগ্রামাররা একটি কাজ একাধিকবার করতে পছন্দ করেন না। এই কারণে প্রোগ্রামের ভাষায় for-loop হলো অন্যতম একটি প্রয়োজনীয় স্টেটমেন্ট।

সতর্কীকরণ !!!

যদি তুমি উদাহরণগুলো নিজে করার চেষ্টা করো, তাহলে তোমার for-loop এর কোড টাইপ করার সময় কিছু হাস্যকর ত্রুটিযুক্ত বার্তা পাবে। যদি তুমি করো, সেটি হতে পারে এমন:

IndentationError: expected an indented block

এরূপ ত্রুটি দেখতে পেলে তোমাকে বুঝতে হবে দ্বিতীয় লাইনে ফাঁকা স্থান দিতে ভুল করেছো। পাইথনে ফাঁকা স্থান (সাধারণ স্পেস কিংবা ট্যাব) খুবই গুরুত্বপূর্ণ। আমরা একটু পরেই এই বিষয়ে বিস্তারিত আলোচনা করবো...

range ব্যবহারে আমাদের এতোটা কঠোর না হলেও চলবে। আমরা list (লিস্ট) ও ব্যবহার করেছি যা ইতোমধ্যেই আমরা তৈরি করেছি। যেমন পূর্বের অধ্যায়ে আমরা তৈরি করেছিলাম কেনাকাটার তালিকা:

```
>>> shopping_list = ['eggs, milk, fish, banana, potato, meat']
>>> for i in shopping_list:
... print(i)
eggs
milk
fish
banana
potato
meat
```

উপরের কোডটি থেকে বলা যায়, "list" এর প্রতিটি আইটেমের জন্য, ভ্যারিয়েবল 'i' এর মধ্যে মান সংরক্ষণ করো এবং তারপর উক্ত ভ্যারিয়েবলের বিষয়বস্তুসমূহ print করো। আমি আবার বলছি, যদি আমরা for-loop থেকে অব্যাহতি পেতে চাই, তাহলে আমাদের এ ধরনের কিছু ব্যবহার করতে হবে।

```
>>> shopping_list = ['eggs, milk, fish, banana, potato, meat']

>>> print(shopping_list[0])
eggs
>>> print(shopping_list[1])
milk
>>> print(shopping_list[2])
 fish
>>> print(shopping_list[3])
 banana
>>> print(shopping_list[4])
 potato
>>> print(shopping_list[5])
 meat
```

সুতরাং আবারো বলতে চাই, loop (লুপ) আমাদেরকে একই কথা এবং একই লেখা বারবার টাইপ করা থেকে রক্ষা করে।

৫.১ কখন একটি ব্লক বর্গক্ষেত্র নয় ?

যখন এটি একটি কোড ব্লক
সুতরাং এখন 'কোড ব্লক' কী ?
একটি কোড ব্লক হলো প্রোগ্রামিং স্টেটমেন্টের একটি সেট যাকে আমরা একটি গ্রুপে বা দলে পরিণত করতে চাই। উদাহরণস্বরূপ বলা যায়, উপরের for-loop উদাহরণটিতে, তুমি আইটেমগুলো শুধুমাত্র print করা ছাড়াও আরো কিছু করতে চাও। হয়তো তুমি চাও যে প্রতিটি আইটেম ক্রয় করবে এরপর তুমি print করবে যে আইটেমটি কি ছিলো। ধরে নাও আমাদের একটি 'buy' নামের ফাংশন রয়েছে, এবার তুমি এভাবে কোডটি লিখতে পারো:

```
>>> for i in shopping_list:
... buy(i)
... print(i)
```

পাইথন কনসোলে উপরের উদাহরণ লিখে ভয় পেয়ো না, কেননা আমাদের কোন ফাংশন নেই এবং যদি কোডটি তুমি () চালাতে চেষ্টা করো তাহলে তুমি একটি ত্রুটি বার্তা (error message) পাবে। কিন্তু এই উদাহরণটি একটি সাধারণ পাইথন ব্লক প্রদর্শন করে যেটি ২টি কমান্ডের সমন্বয়ে গঠিত:

```
buy(i)
print(i)
```

পাইথনে ফাঁকা স্পেস যেমন ট্যাব (যখন তুমি ট্যাব কী চাপ দাও) এবং স্পেস (যখন তুমি স্পেসবার কী চাপ দাও) খুবই গুরুত্বপূর্ণ। কোড যেগুলো একই অবস্থানে রয়েছে সেগুলোকে ব্লক তৈরির জন্য একত্রে গ্রুপ করা হয়।

```
this would be block 1
this would be block 1
this would be block 1
     this would be block 2
     this would be block 2
     this would be block 2
this would still be block 1
this would still be block 1
     this would be block 3
     this would be block 3
          this would be block 4
          this would be block 4
          this would be block 4
```

কিন্তু তোমাকে অবশ্যই ফাঁকা জায়গা (স্পেসিং) বিষয়ক নিয়ম-রীতি অনুসরণ করতে হবে। উদাহরণস্বরূপ:

```
>>> for i in shopping_list:
...     buy(i)
...       print(i)
```

এখানে দ্বিতীয় লাইনে (ফাংশন buy(i)) শুরু হয়েছে ৪টি স্পেস দ্বারা। তৃতীয় লাইনে (print(i)) শুরু হয়েছে ৬টি স্পেস দ্বারা। চলো এবার আমরা আগের কোডটি নতুন করে ফাঁকা স্থান (স্পেস) দৃশ্যমান অবস্থায় দেখি (স্পেস বুঝাতে আবারো @ ব্যবহৃত হয়েছে):

```
>>> for i in shopping_list:
... @@@@buy(i)
... @@@@@@print(i)
```

এটি একটি ত্রুটি দেখাবে। একবার তুমি যদি ৪টি স্পেস ব্যবহার করো, তবে তোমাকে এই ৪টি স্পেস চালিয়ে যেতে হবে। এবার যদি তুমি একটি ব্লকের অভ্যন্তরে আরেকটি ব্লক রাখতে চাও, সেক্ষেত্রে ঐ অভ্যন্তরীণ ব্লকের জন্য লাইনের শুরুতে তোমার ৮টি স্পেস (২ x ৪) ব্যবহার করতে হবে।

সুতরাং প্রথম ব্লকে রয়েছে ৪টি স্পেস (আমরা পুনরায় সেগুলো হাইলাইট করছি যাতে করে তোমরা দেখতে পারো):

```
@@@@here's my first block
@@@@here's my first block
```

এবং অতঃপর দ্বিতীয় ব্লকের (যেটি প্রথমটির 'অভ্যন্তর' -এ রয়েছে) তার প্রয়োজন ৮টি স্পেস:

```
@@@@here's my first block
@@@@here's my first block
@@@@@@@@here's my second block
@@@@@@@@here's my second block
```

কিন্তু কেন আমরা একটি ব্লকের অভ্যন্তরে অন্য একটি ব্লক রাখতে চাই? সাধারণত এই কাজ আমার তখনই করি যখন দ্বিতীয় ব্লকটি কোন না কোনভাবে প্রথমটির উপর নির্ভর করে। যেমনটি আমাদের for-loop। যদি লাইনসহ for-loop টি প্রথম ব্লক হয়, তখন স্টেটমেন্টগুলো যা আমরা একের পর এক চালাতে চাই দ্বিতীয় ব্লকে এমনভাবে থাকে যেন তারা প্রথম ব্লকটির উপর নির্ভর করে সঠিকভাবে কাজ করে।
পাইথন কনসোলে, একবার তুমি কোড লেখা শুরু করলে, পাইথন উক্ত ব্লকটি ততক্ষণ পর্যন্ত চালিয়ে যাবে যতক্ষণ না তুমি ফাঁকা লাইনে এন্টার কী-তে চাপ দিচ্ছো (লাইনের শুরুতে তুমি ৩টি ডট দেখতে পাবে যা তোমাকে দেখাবে যে তুমি এখনো ব্লকের মধ্যে রয়েছো)।

চলো বাস্তব কিছু উদাহরণ নিজেরা করার চেষ্টা করি। পাইথন কনসল খুলো এবং নিম্নলিখিত কোডটি লিখো (মনে রেখো যে print লাইনের শুরুতে তোমার ৪বার স্পেসবারটি চাপার প্রয়োজন পড়বে):

```
>>> mylist = [ 'a', 'b', 'c' ]
>>> for i in mylist:
... print(i)
... print(i)
...
a
a
b
b
c
c
```

দ্বিতীয় print এর পরে, ফাঁকা লাইনে এন্টার কী চাপো- যেটি কনসোলকে জানাবে যে তুমি ব্লকটি শেষ করতে চাও। তারপর এটি লিস্টের প্রতিটি আইটেম দুইবার প্রিন্ট করবে।
পরবর্তী উদাহরণটিতে একটি ত্রুটি বার্তা প্রদর্শন করবে:

```
>>>mylist = [ 'a', 'b', 'c' ]
>>> for i in mylist:
... print(i)
... print(i)
...
File <stdin>, line 3
print(i)
^
IndentationError: unexpected indent
```

দ্বিতীয় প্রিন্ট লাইনে ৪টি নয়, ৬টি স্পেস রয়েছে, যেটি পাইথন মোটেও সমর্থন করে না, কারণ সে একই অবস্থানে থাকার জন্য এর জায়গায় থেকে শুরু করা প্রয়োজন।

মনে রেখো

তুমি যদি ৪টি স্পেস ব্যবহার করে কোড ব্লক শুরু করো তাহলে তোমাকে এই ৪টি স্পেস চালিয়ে যেতে হবে। আর যদি ২টি স্পেস দিয়ে ব্লক শুরু করো তাহলে তোমাকে অবশ্যই ২টি স্পেস চালিয়ে যেতে হবে। ৪টি স্পেস ব্যবহারের কারণ হলো বেশিরভাগ মানুষ এটি ব্যবহার করে।

নিচের উদাহরণটি ২ ব্লক কোডের একটু জটিল উদাহরণ:

```
>>> mylist = [ 'a', 'b', 'c' ]
>>> for i in mylist:
... print(i)
... for j in mylist:
... print(j)
...
```

এই কোডের ব্লকগুলো কোথায় এবং কী হতে চলেছে এখানে...?
এই কোডে দুইটি ব্লক রয়েছে- প্রথমটি হলো প্রথম **for-loop** এর অংশ:

```
>>> mylist = [ 'a', 'b', 'c' ]
>>> for i in mylist:
... print(i)              #
... for j in mylist:          #-- এই লাইনগুলো প্রথম ব্লকের
... print(j)        #
...
```

দ্বিতীয় ব্লকটি হলো দ্বিতীয় **for-loop** এর একমাত্র প্রিন্ট লাইনটি:

```
>>> mylist = [ 'a', 'b', 'c' ]
>>> for i in mylist:
... print(i)
... for j in mylist:
... print(j)    # এই লাইনটি দ্বিতীয় ব্লকের
...
```

তুমি কী বুঝতে পারছো এই ক্ষুদ্র পরিসরের এই কোডটি কী করতে চলেছে?
এটি 'mylist' থেকে ৩টি অক্ষর প্রিন্ট করতে চলেছে, কিন্তু এটি কতবার প্রিন্ট করবে? যদি আমরা প্রতিটি লাইনের দিকে লক্ষ করি, তাহলে সম্ভবত সংখ্যাটি বের করতে পারবো। আমরা জানি যে প্রথম লুপ স্টেটমেন্টটি লিস্টের প্রতিটি আইটেম পড়বে এবং তারপর ১ নম্বর ব্লকে কমান্ড চালাবে। সুতরাং এটি একটি অক্ষর প্রিন্ট করবে, এরপর পরের লুপটি শুরু। এই লুপটিও লিস্টের প্রতিটি আইটেম পড়বে এবং তারপর ২ নম্বর ব্লকে কমান্ড চালাবে। সুতরাং এই কোডটি চললে আমাদের যা পাওয়া উচিৎ, তাহলো '**a**' এর পর আসবে 'a', 'b', 'c', তারপর 'b' এর পর আসবে 'a', 'b', 'c', ইত্যাদি। পাইথন কনসোলে কোডটি লিখে তোমারা নিজেরা দেখো:

```
>>> mylist = [ 'a', 'b', 'c' ]
>>> for i in mylist:
... print(i)
... for j in mylist:
```

```
... print(j)
...
a
a
b
c
b
a
b
c
c
a
b
c
```

শুধুমাত্র অক্ষর প্রিন্ট অপেক্ষা বেশি গুরুত্বপূর্ণ কিছু ব্যবহার করলে কেমন হয়? তোমাদের নিশ্চয়ই মনে আছে, এই বই-এর শুরুতে আমরা একটি হিসেব করেছিলাম। আর সেটি ছিলো প্রতি বছর তোমার কত টাকা জমবে। তোমাদের তা মনে আছে, মনে না থাকলেও সমস্যা নেই। চলো আমরা আবার হিসেবটা মনে করি। প্রতি মাসে টিফিনের টাকার জন্য তুমি পেতে ২০০ টাকা, বৃত্তির জন্য পেতে ৩০০ টাকা এবং প্রতি মাসে তোমার খরচ হলো ২৫০ টাকা। এই হিসেব থেকে আমরা বের করেছিলাম তোমার বছরে কত টাকা জমা হবে। তাহলে হিসাবটি হবে নিম্নরূপ:

```
>>> ( 200+300-250)*12
```

(এই ৳200 + ৳300 - ৳250 কে বছরের 12 মাস দ্বারা গুণ করতে হবে।)
প্রতি মাসে কী করলে তোমার সঞ্চয় বাড়ছে? বাড়লে তা কীভাবে বাড়ছে? সেটি আমাদের আগে লক্ষ করা উচিৎ। আমরা আরেকটি for-loop দ্বারা এই বাৎসরিক হিসাবের কাজটি করতে পারি। কিন্তু এর জন্য প্রথমে আমাদের ঐ সংখ্যাগুলোকে ভ্যারিয়েবলে হিসেবে উল্লেখ করতে নিতে হবে:

```
>>> tiffin = 200
>>> stipend = 300
>>> spending = 250
```

আমরা মূল হিসাবে ভ্যারিয়েবলগুলো ব্যবহার করে করতে পারি:

```
>>> (tiffin+stipend-spending)*12
3000
```

অথবা যেহেতু আমরা দেখতে পারছি প্রতি মাসেই সঞ্চয় বৃদ্ধি পাচ্ছে, এজন্য savings নামের নতুন আরেকটি ভ্যারিয়েবল তৈরি করে এরপর loop ব্যবহার করতে হবে:

```
1.>>> savings=0
2.>>> tiffin=200
3.>>> stipend=300
4.>>> spending=250
5.>>> for month in range(1,13):
      savings=savings+tiffin+stipend-spending
      print('month %s=%s'%(month, savings))
```

প্রথম লাইনে 'savings' ভ্যারিয়েবল লোড করা হয়েছে 0 দ্বারা (কারণ আমার এখনো শুরুতে কোন কিছুই সংরক্ষণ করা থাকবে না)।
দ্বিতীয় লাইনটি for-loop দ্বারা গঠিত হয়েছে, যা ব্লকের (৩ ও ৪ নং লাইন দ্বারা ব্লক গঠিত হয়েছে) কমান্ডগুলো দিয়ে চালিত হবে। যত বার এই লুপ চলবে, month ভ্যারিয়েবল রেঞ্জের পরবর্তী সংখ্যার সাথে লোড হবে।
তৃতীয় লাইনটি একটু বেশি জটিল। সাধারণত, প্রতি মাসে আমরা চাই যেটি আমরা সঞ্চয় করলাম সেটি মোট সঞ্চয়ের সাথে যোগ করতে। একটু চিন্তা করলেই দেখবে 'savings' ভ্যারিয়েবলের সাথে ব্যাংকের মিল আছে।
আমরা টিফিনের টাকা এবং বৃত্তি থেকে পাওয়া টাকার যোগফল থেকে, প্রতি মাসে খরচ করা টাকা বিয়োগ করে বাকী টাকা ব্যাংকে রাখি। সুতরাং কম্পিউটারের ভাষায়, তৃতীয় লাইনের মানে হলো, "আমার বর্তমান সঞ্চয় দ্বারা savings ভ্যারিয়েবলের বিষয়বস্তুকে প্রতিস্থাপন করা, এবং এই মাসে যা সঞ্চয় হলো তা পরবর্তী মাসের সাথে যোগ করা"। সাধারণত, সমান চিহ্নটি (=) একটি বিশেষ কোডের টুকরো যা নির্দেশে দেয়, "ডান পাশের উপাদানগুলোর সমাধান বা কাজ আগে করতে এবং তারপর তার বাম পাশে নাম ব্যবহার করে ভবিষ্যতে ব্যবহারের জন্য তা সংরক্ষণ করতে"।
চতুর্থ লাইনটি একটু জটিল print স্টেটমেন্ট, যেটি মাসের সংখ্যা এবং ঐ মাসে মোট জমাকৃত টাকার পরিমাণ কম্পিউটারের পর্দায় প্রিন্ট করে। যদি এই লাইনের অর্থ তোমার কাছে কঠিন মনে হয়ে থাকে তাহলে, ১৮ পৃষ্ঠার *Tricks with Strings* সেকশনটির সাথে মিলিয়ে দেখো। সুতরাং, এই প্রোগ্রামটি তুমি চালালে নিম্নলিখিত বিষয়টি দেখতে পাবে:

```
Month 1 = 250
Month 2 = 500
Month 3 = 750
Month 4 = 1000
Month 5 = 1250
Month 6 = 1500
Month 7 = 1750
Month 8 = 2000
Month 9 = 2250
Month 10 = 2500
Month 11 = 2750
Month 12 = 3000
```

..... এভাবে লেখা ১২ মাস পর্যন্ত চলতে থাকবে।

৫.২ যেহেতু আমরা লুপ নিয়ে আলোচনা করছি

পাইথনে for-loop শুধু একমাত্র লুপ নয়। পাইথনে while-loop ও রয়েছে। যদি for-loop এমন একটি লুপ হয় যে প্রোগ্রামে কখন তোমাকে থামতে হবে, তাহলে while-loop হলো এমন একটি লুপ যেখানে তোমার জানার প্রয়োজন নেই কখন তোমাকে থামতে হবে। মনে করো, একটি সিঁড়িতে ২০টি ধাপ রয়েছে। তুমি জানো সহজেই তুমি ২০টি ধাপ অতিক্রম করতে পারো। এটিই হলো for-loop।

```
>>> for step in range(0,20):
... print(step)
```

এবার মনে করো এমন একটি সিঁড়ির কথা যা পাহাড়ের গা বেয়ে উঠেছে। পাহাড়ের চূঁড়ায় ওঠার আগেই তোমার শক্তি শেষ হতে পারে। অথবা আবহাওয়ার বিরূপ প্রতিক্রিয়া তোমাকে পর্বত আহরণ বন্ধ করার জন্য বাধ্য করতে পারে। এটিই হলো while-loop।

```
>>> step = 0
>>> while step < 10000:
... print(step)
... if tired == True:
... break
... elif badweather == True:
... break
... else:
... step = step + 1
```

উপরিউক্ত কোডটি চালাতে গিয়ে ভয় পেয়ো না, কারণ আমরা আমাদের tired এবং badweather ভ্যারিয়েবল তৈরি নিয়ে চিন্তিত নই। কিন্তু উপরের কোডটি আসলে while-loop এর প্রাথমিক ধারণা প্রকাশ করে। while লুপ ভ্যারিয়েবল step 1000 অপেক্ষা কম (step < 10000) ব্লকের কোড এক্সিকিউট করে। ব্লকে, প্রথমে আমরা step এর মান প্রিন্ট করি, এরপর পরীক্ষা করি tired অথবা badweather এর মধ্যে কোনটি সত্য। যদি tired সত্য হয়, তাহলে break স্টেটমেন্ট লুপে কোড এক্সিকিউট করা বন্ধ করে দেয় (মূলতঃ আমরা কোড থেকে লাফ দিয়ে লুপের বাইরে ব্লকের ঠিক পরের লাইনে প্রবেশ করি)। badweather সত্য হলেও একই ঘটনা ঘটবে। যদি tired অথবা badweather সত্য না হয় অর্থ্যাৎ মিথ্যা হয়, তাহলে step মানে 1 যুক্ত হবে, এবং while loop (step < 10000) এর শর্তগুলো পুনরায় পরীক্ষা হবে।
সুতরাং while loop এর ধাপগুলো হলো:

- শর্ত পরীক্ষা করা
- ব্লকের কোড এক্সিকিউট করা
- পুনরায় শুরু করা

আরো সাধারণভাবে বলতে গেলে, while-loop একের অধিক শর্ত দিয়েও গঠিত হতে পারে। যেমন-

```
>>> x = 45
>>> y = 80
>>> while x < 50 and y < 100:
... x = x + 1
... y = y + 1
... print(x, y)
```

উপরের লুপটিতে, আমরা একটি ভ্যারিয়েবল x নিয়েছি যার মান 45, এবং ভ্যারিয়েবল y নিয়েছি যার মান 80। লুপটিতে একসাথে দুইটি শর্ত পরীক্ষা কর বলতে বোঝানো হচ্ছে: x, 50 অপেক্ষা ছোট কিনা এবং y, 100 অপেক্ষা ছোট কিনা। যখন দুইটি শর্তই সত্য, তখন কোড ব্লকটি এক্সিকিউট করবে, প্রত্যেক (x এবং y) ভ্যারিয়েবলে 1 করে যোগ হবে এবং তাদের মান প্রিন্ট হবে। কোডটির আউটপুট হবে এরূপ:

```
46 81
47 82
48 83
49 84
50 85
```

এখন হয়তো নিশ্চয় তোমরা খুঁজে বের করতে পারবে কেন উপরের সংখ্যাগুলো প্রিন্ট হলো?[১]
While-loop এর আরেকটি সাধারণ ব্যবহার হলো, এটি একটি অর্ধ-চলমান লুপ। মূলতঃ কোডে কোন প্রকার পরিবর্তন ঘটা ছাড়া এটি অবিরাম চলতে পারে। উদাহরণটি দেখো-

```
>>> while True:
... lots of code here
... lots of code here
... lots of code here
... if some_condition == True:
... break
```

উপরের কোডে while-loop এর শর্ত শুধুমাত্র 'True'। এজন্য এটি ব্লকে কোড চালাবে সবসময় (এভাবেই লুপটি হলো অবিরাম অথবা অনন্ত)। শুধুমাত্র যদি ভ্যারিয়েবল some condition সত্য হলে কোড লুপ থেকে বের হয়ে যাবে। তোমরা এই ধরনের শর্তযুক্ত while-loop চমৎকার উদাহরণ পরিশিষ্ট C-তে দেখতে পাবে (এই সেকশনটি র‍্যানডম মডিউলের সম্পর্কে), কিন্তু তোমার সেই দৃষ্টি ফেরানোর পূর্বে পরবর্তী অধ্যায় না পড়া পর্যন্ত অপেক্ষা করতে হবে।

[১] (ভ্যারিয়েবল x-এর মান 45 এবং ভ্যারিয়েবল y-এর মান 80 থাকা অবস্থায় আমরা গণনা করে শুরু করেছিলাম, এবং অতঃপর প্রত্যেকবার লুপের কোড রান (চলেছে) করায় প্রতিটি ভ্যারিয়েবলের নাম 1 করে বৃদ্ধি পেয়েছে। শর্তগুলো পরীক্ষা করে যে x, 50 অপেক্ষা অবশ্যই, এবং y, 100 অপেক্ষা অবশ্যই ছোট কিনা। পাঁচবার লুপটি চলার পর (প্রতিবার প্রত্যেক ভ্যারিয়েবলে 1 যোগ হয়েছে) x এর মান হয় 50। সুতরাং এখন প্রথম শর্তটি (x < 50) এখন আর সত্য নয়, সুতরাং এজন্য পাইথন জানে যে লুপটি বন্ধ করতে হবে।)

৫.৩ নিজেরা একটু চেষ্টা করো

এই অধ্যায়ে আমরা দেখলাম একটি কাজ বারবার করার জন্য কীভাবে লুপ ব্যবহার করতে হয়। আমরা লুপের ভিতরে ব্লক কোড ব্যবহার করেছি পুনরায় কাজটি করার জন্য।

অনুশীলনী ১

নিম্নলিখিত কোডটি চালালে কী ঘটবে বল তো?

```
>>>for x in range(0, 20):
...print('hello %s' % x)
...if x < 9:
…   break
```

অনুশীলনী ২

যখন তুমি ব্যাংকে কোন টাকা জমা করো, তখন ব্যাংক তোমাকে সুদ দেয়। সুদ হলো সেই অর্থ যা ব্যাংক তোমার টাকা ব্যবহার করার জন্য তোমাকে কিছু অর্থ প্রদান করে, প্রতি বছর ব্যাংক তোমার জমাকৃত অর্থের উপর ভিত্তি করে এই সুদ প্রদান করে। এই অর্থ সাধারণত তোমার ব্যাংকে সঞ্চিত টাকার সাথে যুক্ত হয়ে যায়, এবং এটিও তোমাকে বাড়তি টাকা প্রদান করবে...কিন্তু কীভাবে? এই ধারণাটি পাওয়ার জন্য তুমি তোমার মা, বাবা কিংবা বড় করো কাছ থেকে জেনে নিতে পারো। সুদ নির্ণয় করার জন্য শতকরা ব্যবহার হয়। তুমি যদি না জানো যে শতকরা কি, চিন্তা করো না। মনে করো ব্যাংক তোমাকে জমাকৃত টাকার (জমা করেছো 1000 টাকা) জন্য 1% (শতকরা 1 ভাগ) সুদ প্রদান করছে, তাহলে তুমি জমাকৃত টাকার সাথে 0.01 গুণ করো (হিসাবটি হলো 1000 X 0.01 টাকা সুদ প্রদান করছে)। যদি শতকরা 2 ভাগ সুদ প্রদান করছে, তাহলে তুমি জমাকৃত টাকার সাথে 0.02 গুণ করতে হবে। যদি তুমি 100 টাকা ব্যাংকে জমা করো, ব্যাংক যদি তোমাকে প্রতি বছর 3% সুদ প্রদান করে, তাহলে প্রতি বছর তোমার টাকার পরিমাণ কত হবে, এবং এভাবে 10 বছরে কত হবে? এটি নির্ণয় করার জন্য তোমাকে for-loop ব্যবহার করতে হবে। তাহলে এই প্রোগ্রামটি লিখতে হবে (মনে রাখতে হবে মোট টাকার সাথে সুদ যোগ করতে হবে)।

ষষ্ঠ অধ্যায়

অব্যবহৃত ? তাহলে নিশ্চয়ই পুনরায় ব্যবহারযোগ্য !!

একটু চিন্তা করে দেখ তো প্রতিদিন তুমি কত কত আবর্জনা তৈরি করো! পানি অথবা জুসের বোতল, চিপসের প্যাকেট, স্যান্ডউইচ মোড়ানো প্লাস্টিক, ফাস্টফুড খাবারের পর প্লাস্টিকের ট্রে, ফ্রাইড চিকেনের প্যাকেট, প্লাস্টিকের শপিং ব্যাগ, বাজারের ব্যাগ, সংবাদ পত্রিকা, ম্যাগাজিন ইত্যাদি আরও অনেক কিছু।

এবার কল্পনা করো এই সব আবর্জনা তোমার বাড়ির উঠোনে স্তূপ করে রাখা হলো। কেমন হবে বলো তো?

সাধারণত যেটা কল্পনা করা যায় সেটা হলো স্কুলে যাবার পথে দুষ্ট বাচ্চারা এই ময়লার স্তূপের চূড়ায় উঠে নাচানাচি করবে! খুব ভাগ্যবান হলে এই ঘটনার ব্যতিক্রম হবে। অবশ্য আমার ধারণা তুমি আবর্জনাগুলোকে এভাবে স্তূপ করে রেখে দেওয়ার চেয়ে, আবার তা কোনভাবে ব্যবহার করা যায় কিনা সেটি চিন্তা করবে। আর চিন্তার কারণেই আধা ভাঙ্গা কাঁচের বোতলগুলো গলিয়ে নতুন জার ও বোতল তৈরি হবে, কাগজগুলো পুনরায় ব্যবহারের জন্য মণ্ডে পরিণত হবে আর প্লাস্টিকের টুকরোগুলো থেকে নতুন কোন জিনিস প্রস্তুত হবে। কাজেই তোমার বাড়ির উঠোন হাজার হাজার আবর্জনার স্তূপের নিচে চাপা পড়ে অদৃশ্য হয়ে যাবেনা। পৃথিবী পৃষ্ঠের একটি অংশ বেদখল করার চেয়ে এটি ভালো নয় কি? হ্যাঁ, আমাদেরকে একই জিনিস পুনরায় ব্যবহারের অভ্যাস করতে হবে। তা না হলে অব্যবহৃত বস্তুর স্তূপের পাহাড় জমে এ পৃথিবী তার বাসযোগ্যতা হারাবে।
প্রোগ্রামিং-এর দুনিয়ায় একই কোড পুনরায় এমনকি বারবার ব্যবহারের চর্চা করা হয়। দুই ধরনের চিন্তা থেকে এটি করা হয়।

এক. সৃষ্ট প্রোগ্রাম ব্যবহার না করলে স্তূপ জমে একে অপরের নিচে চাপা পড়ে যাবে
দুই. একই কোড বার বার লিখে সময় ও শ্রমের অপচয় হবে

বেশ কয়েকটি উপায়ে পাইথন ও সাধারণ যেকোন প্রোগ্রামিং ভাষার কোড পুনরায় ব্যবহার করা যায়। ইতোমধ্যে আমরা একটি উপায় দেখেছি অধ্যায়-৩ এ, ফাংশন রেঞ্জ-এর মাধ্যমে। অর্থাৎ একবার একটি কোড লিখে ফেললে কোন প্রোগ্রামে সেই কোড বারবার ব্যবহার করা যায়। খুব সাধারণ একটি ফাংশনের উদাহরণ দিয়ে আমরা বিষয়টি বোঝার চেষ্টা করি:

```
>>> def myfunction(myname):
...   print('hello %s' % myname)
...
```

উপরের উদাহরণটিতে ফাংশনের নাম হচ্ছে myfunction এবং এর প্যারামিটার myname. এখানে প্যারামিটার মূলত এক ধরনের চলক যা শুধু ফাংশনের বডির মধ্যে থাকে। দেখা যাচ্ছে, def অংশটির ঠিক পর পরই ফাংশনের নাম ও তার প্যারামিটার দিয়ে কোডটি ব্লক করে দেয়া হয়েছে (def দেখে চমকে যেও না, এটি আসলে definition এর সংক্ষিপ্ত রূপ)। আবার, বন্ধনীতে আবদ্ধ প্যারামিটারের মান দিয়েও ফাংশনটি চালু করা যায়। যেমন:

```
>>> myfunction('Nipun')
hello Nipun
```

দুটি প্যারামিটার নিয়েও কাজ করা যায়:

```
>>> def myfunction(firstname, lastname):
...     print('Hello %s %s' % (firstname, lastname))
...
```

প্রথম ফাংশনের মতো করেই বলা যায়:

```
>>> myfunction('Nipun', 'Ahmed')
Hello Nipun Ahmed
```

অথবা আমরা নিজেরাই কিছু চলক তৈরি করে ফাংশনে ব্যবহার করতে পারি:

```
>>> fn = 'Asif'
>>> ln = 'Tanvir'
>>> myfunction(fn, ln)
Hello Asif Tanvir
```

স্টেটমেন্ট রিটার্ন (return statement) করার মাধ্যমে আমরা যে কোন ফাংশনের মান পুনরায় ফেরত পেতে পারি:

```
>>> def savings(tiffin + stipend - spending):
...           return  tiffin + stipend - spending
...
>>> print(savings(200, 300, 250))
250
```

উপরোক্ত ফাংশনে তিনটি প্যারামিটার বিদ্যমান। প্রথম দুটি প্যারামিটার (**tiffin** এবং **stipend**) যোগ করে যোগফল থেকে তৃতীয় প্যারামিটার (spending) বিয়োগ করা হলো। প্রাপ্ত এই ফলাফলকে অন্য একটি চলকের মান

হিসেবে ব্যবহার করা যেতে পারে। অন্যান্য চলক-এ যেভাবে মান বসানো হয় একই উপায়ে এই মান ব্যবহার করা যেতে পারে।

```
>>> my_savings = savings(20, 10, 5)
>>> print(my_savings)
25
```

আর একটি ব্যাপার হচ্ছে ফাংশনের মূল অংশে (body) ব্যবহৃত চলক ফাংশন শেষে ব্যবহারযোগ্য নয়:

```
>>> def variable_test():
...             a = 10
...             b = 20
...             return a * b
...
>>> print(variable_test())
200
>>> print(a)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

উপরের উদাহরণটিতে আমরা variable_test ফাশন তৈরি করেছি। এখানে দুটি চলক (a ও b) গুণ করা হয়েছে। ফাংশন এ print করতে বললে প্রাপ্ত ফলাফল পাওয়া যাবে 200। এরপর যদি a অথবা b এর মান print করতে বলা হয় তাহলে একটি ত্রুটি বার্তা দেখাবে যে, " 'a' is not defined" অর্থাৎ 'a' সংজ্ঞায়িত নয়। প্রোগ্রামিং এর ভাষায় একে বলা হয় 'scope'।

ফাংশনকে সমুদ্রে ভাসমান এক টুকরা দ্বীপ হিসেবে কল্পনা করতে পারো যেখান থেকে অন্য কোথাও সাঁতরে পার হওয়া কল্পনাতীত। কখনো কখনো দ্বীপের উপর দিয়ে প্লেন উড়ে যাওয়ার সময় কাগজের টুকরো ফেলে যেতে দেখা যায় (এই কাগজের টুকরোগুলোকে ফাংশনের প্যারামিটার হিসেবে কল্পনা করতে পারো)। ধরে নাও, দ্বীপের অধিবাসীরা প্রতিটি কাগজের টুকরোকে একেকটি বার্তা মনে করে বোতল ভর্তি করে রাখে। একসময় বোতলটি সমুদ্রে ফেলে দেয় (মনে করো, এটি মান রিটার্ন করা বা return value)। এ ঘটনায় কতজন অধিবাসী বার্তা সঞ্চয় করলো আর কতজন তা পড়ে সমুদ্রে ফেলে দিলো তাতে কিছুই যায় আসে না। এ থেকে আমরা 'scope' এর ধারণা পেতে পারি। কিন্তু ঘটনাটিতে যদি এমন হয় যে দ্বীপের একজন অধিবাসীর নিকট একটি বাইনোকুলার আছে এবং প্রধান ভূমিতে অবস্থিত লোকজনের কাজকর্ম ও ঐ ভূমিতে যাওয়ার সবগুলো রাস্তা সে বাইনোকুলার ব্যবহার করে দেখতে পারে তাহলে যা হবে তা হচ্ছে:

```
>>> x = 100
>>> def variable_test2():
...             a = 10
...             b = 20
...             return a * b * x
...
>>> print(variable_test2())
20000
```

অর্থাৎ, যদিও a ও b চলক ফাংশনের বাইরে অব্যবহারযোগ্য এবং ফাংশনের বাইরে সৃষ্ট x চলক ঠিকই ব্যবহারযোগ্য হয়েছে। তার মানে দাঁড়াচ্ছে দ্বীপের অন্য অধিবাসীর তুলনায় বাইনোকুলারধারী ব্যক্তি সবগুলো পথ ও পন্থা দেখে

নিয়ে, যে বার্তাগুলো ব্যবহার হচ্ছে না তাও ব্যবহার করতে পারবে।

এর আগে আমরা for-loop ব্যবহার করে যেভাবে আমরা আমাদের এক বছরের সঞ্চয় বের করেছিলাম তাতে খুব সহজেই আমরা অপর একটি ফাংশনে ব্যবহার করতে পারি:

```
>>> def yearly_savings(tiffin + stipend - spending):
...     savings = 0
...     for month in range(1, 13):
...         savings = savings + tiffin +stipend - spending
...         print('month %s = %s' % (month, savings))
...
```

কনসোলে বিভিন্ন মানের জন্য উপরে উল্লেখিত ফাংশন প্রবেশের চেষ্টা করো:

```
>>> yearly_savings(200, 300, 250)
Month 1 = 250
Month 2 = 500
Month 3 = 750
Month 4 = 1000
Month 5 = 1250
Month 6 = 1500
Month 7 = 1750
Month 8 = 2000
Month 9 = 2250
Month 10 = 2500

(continues on...)
```

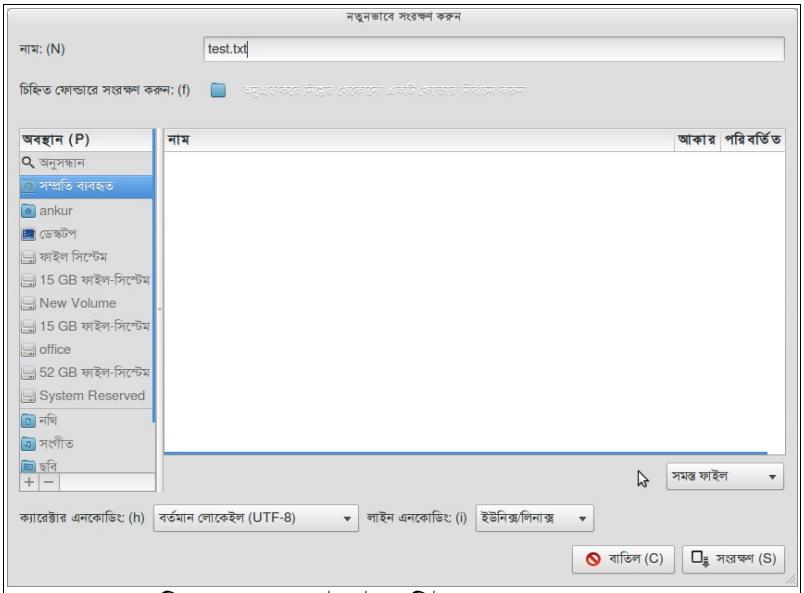
```
>>> yearly_savings(25, 15, 10)
month 1 = 30
month 2 = 60
month 3 = 90
month 4 = 120
month 5 = 150

(continues on...)
```

এই পদ্ধতি for-loop এ প্রতিবার ভিন্ন ভিন্ন মান বসানোর সময় পুনরায় টাইপ করা থেকে মুক্তি দেয়। ফাংশন যখন দলগতভাবে কোথাও ব্যবহৃত হয় তখন তাকে বলে মডিউল। পাইথনে মডিউল খুবই গুরুত্বপূর্ণ।

পরে আমরা মডিউল সম্পর্কে আরও জানবো ...

৬.১ বিট ও পিস (Bits and Pieces)

কম্পিউটারে পাইথন ইনস্টলেশনের সময় স্বয়ংক্রিয়ভাবে প্রয়োজনীয় ফাংশন ও মডিউল ইনস্টল হয়। এদের মধ্যে রেঞ্জ ফাংশন আমরা ইতোমধ্যে ব্যবহার করেছি যদিও অন্য ফাংশন নিয়ে এখনও কাজ করিনি, তাই না?



চিত্র ৬.১ gedit টেক্সট এডিটর এ save ডায়ালগ

ফাংশনে ফাইলের ব্যবহার সম্পর্কে জানতে একটি টেক্সট এডিটর খোল। সেখানে কিছু একটা লিখে File এ ক্লিক করে Save অপশনে গিয়ে যেকোন একটি ফোল্ডার যেমন: Desktop নির্বাচন করো। ফাইলের নাম দাও 'test.txt' (চিত্র ৬.১)।
পাইথন কনসোল খুলে লিখ:

```
>>> f = open('Desktop/test.txt')
>>> print(f.read())
```

লক্ষ করি, ছোট্ট এই কোড দ্বারা কি বোঝানো হচ্ছে। প্রথম লাইনটি হচ্ছে ফাংশনের ফাইল। এতে প্যারামিটার হিসেবে এইমাত্র যে ফাইলটি তুমি তৈরি করেছ তার নাম দেওয়া হয়েছে। এই ফাংশনটি একটি বিশেষ মান (value) রিটার্ন (return) করবে যা উল্লেখিত ফাইলকে উপস্থাপন করে। এই বিশেষ মানকে বলে অবজেক্ট (Object) বলা হয়।

এখানে প্যারামিটার দ্বারা সরাসরি উক্ত ফাইল বোঝানো হচ্ছে না বরং ফাইলের অবস্থান নির্দেশ করা হচ্ছে। অর্থাৎ এই ফাইলের অবজেক্ট f চলক-এর মধ্যে সংরক্ষিত।

পরবর্তী লাইনটি ফাইল অবজেক্টের অন্য একটি বিশেষ ফাংশন (read)-কে কল (call) করেছে। এই ফাংশনের সাহায্যে ফাইলের বিষয়বস্তু পড়ে কনসোলে তা মুদ্রণের নির্দেশ দেওয়া হচ্ছে। এখানে দেখা যাচ্ছে f চলক-এ বিদ্যমান অবজেক্ট পড়ার জন্য read ফাংশন ব্যবহার করা হয়েছে এবং f ও read এর মধ্যে ডট (.) চিহ্ন ব্যবহৃত হয়েছে।

বইয়ের শেষে বিদ্যমান **Appendix B** থেকে পাইথনে ফাংশন তৈরি সম্পর্কে আমরা আরও জানতে পারবো।

৬.২ মডিউল (Modules)

ইতোমধ্যে আমরা একটি কোড বার বার ব্যবহারের নানারকম পদ্ধতি সম্পর্কে জেনে ফেলেছি। প্রয়োজনে আমরা ফাংশন নিজেরাই তৈরি করতে পারি অথবা পাইথনে সৃষ্ট ফাংশনও (যেমন: range and file, int and str) ব্যবহার করতে পারি। আর এক ধরনের বিশেষ ফাংশন এবং অবজেক্ট আছে যেগুলোতে ডট (.) প্রতীকের মাধ্যমে অনেক ফাংশন ও অবজেক্ট একত্রিত করে ব্যবহৃত হয় এবং একে বলা হয় মডিউল। 'time' মডিউলের একটি উদাহরণ নিম্নরূপ:

```
>>> import time
```

import কমান্ড দিয়ে পাইথনকে বলা হয়েছে আমরা একটি মডিউলে প্রবেশাধিকার চাই। উপরের উদাহরণটিতে বলা হচ্ছে আমরা time মডিউলে যেতে যাই। ডট (.) প্রতীক ব্যবহার করে ঐ মডিউলে বিদ্যমান যেকোন ফাংশন এবং অবজেক্ট নিয়ে আমরা কাজ করতে পারি:

```
>>> print(time.localtime())
(2006, 11, 10, 23, 49, 1, 4, 314, 1)
```

এখানে দেখা যাচ্ছে, time মডিউলের অভ্যন্তরে localtime একটি ফাংশন। এটি চলমান সময় এবং তারিখ রিটার্ন করে।আর রিটার্ন করার সময় এটি সময়কে পৃথক পৃথক কয়েকটা অংশে ভাগ করে নেয়। অর্থাৎ সময় সে রিটার্ন করে বছর, মাস, দিন, ঘণ্টা, মিনিট, সেকেন্ড, সপ্তাহের কোন দিন, বছরের কোন দিন এইসব বিবরণ সহকারে। এমনকি দিনের আলো সংরক্ষণ (daylight savings) করে সময় হিসেব করা হচ্ছে কিনা তাও এক্ষেত্রে আমরা দেখতে পাবো। যদি দিনের আলো সংরক্ষণ করে হিসেব করা হয় তাহলে ১, সংরক্ষণ না করা হলে ০। প্রতিটি পৃথক অংশ একটি টাপল (Tuple) এ সংরক্ষিত হয় (২য় অধ্যায়ের টাপল ও তালিকা অংশটুকু একবার দেখে নাও)। time মডিউলে মাধ্যমে localtime যে তারিখ এবং সময় আমাদের রিটার্ন করে তা আমরা আমাদের আঞ্চলিক সময় ও তারিখ অনুসারে রূপান্তর করে নিতে পারি। এর জন্য আমাদের একটু সহজ ও বোধগম্য অপর একটি ফাংশন ব্যবহার করতে হবে:

```
>>> t = time.localtime()
>>> print(time.asctime(t))
Sat Nov 18 22:37:15 2006
```

চাইলে আমরা একটি লাইন ব্যবহার করে পুরো কাজটি করতে পারি:

```
>>> print(time.asctime(time.localtime()))
Sat Nov 18 22:37:15 2006
```

ধরা যাক, তুমি চাও যে ফাংশনটি নিজেই ব্যবহারকারীর নিকট একটি মান প্রবেশ করাতে অনুরোধ করবে। print উক্তি ব্যবহার করে তুমি এই কাজটি করতে পারো। এজন্য time মডিউল যেভাবে ইমপোর্ট করেছ ঠিক সেভাবেই 'sys' মডিউল ইমপোর্ট করতে হবে।

```
import sys
```

sys মডিউলে **'stdin'** নামে একটি অবজেক্ট আছে যাকে বলা হয় short for standard input অর্থাৎ মান বসাতে সংক্ষিপ্ত রূপ। stdin-এ রিডলাইন (readline) নামে প্রয়োজনীয় পদ্ধতি বা ফাংশন আছে যা কীবোর্ডে টাইপকৃত শব্দ পড়তে পারে। Enter কী চাপ দেয়া পর্যন্ত যা লেখা হচ্ছে তা রিডলাইন ফাংশন পড়তে থাকে। এসো আমরা পাইথন কনসোলে নিচের কমান্ড লিখে রিডলাইন পরীক্ষা করে দেখি:

```
>>> print(sys.stdin.readline())
```

এরপর যদি তুমি কোন শব্দ টাইপ করো এবং Enter চাপো তাহলে যে শব্দটি টাইপ করেছিলে তা কনসোলে মুদ্রিত হবে। এক মুহূর্তের জন্য পূর্বে লেখা কোড মনে করার চেষ্টা করো এবং if স্টেটমেন্ট ব্যবহার করে তা লিখে ফেলো:

```
if age >= 10 and age <= 13:
     print('you are %s' % age)
else:
     print('huh?')
```

বার বার নতুন করে মান না বসিয়ে ফাংশনটি নিজেই তোমাকে অনুরোধ করবে বয়স লিখতে অর্থাৎ মান বসাতে। তাহলে দেখা গেল একটু আগেই লেখা কোড ফাংশনে রূপ নিল:

```
>>> def your_age(age):
...     if age >= 10 and age <= 13:
...         print('you are %s' % age)
...     else:
...          print('huh?')
...
```

প্যারামিটারের মান হিসেবে একটি নম্বরকে পাস করানো বলা যায়। এটি কাজ করছে কিনা চলো একটু পরীক্ষা করে নেই:

```
>>> your_age(9)
huh?
>>> your_age(10)
you are 10
```

এখন আমরা জানি ফাংশন নিয়ে আমাদের আর কোন সমস্যা হবে না। আমরা খুব সহজেই তাৎক্ষণিকভাবে যে কোন ব্যক্তির বয়স বের করার জন্য ফাংশনটি পরিবর্তন করতে পারবো :

```
>>> def your_age():
...     print('Please enter your age')
...     age = int(sys.stdin.readline())
```

```
...     if age >= 10 and age <= 13:
...        print('you are %s' % age)
...     else:
...         print('huh?')
...
```

এক্ষেত্রে readline ব্যক্তি টেক্সট (text) বা স্ট্রিং (string) হিসেবে যা টাইপ করবে তা রিটার্ন করে। এই টেক্সটকে সংখ্যায় রূপান্তরের জন্য আমাদের int ফাংশন ব্যবহার করতে হবে। এ কারণেই তা if স্টেটমেন্টে সঠিকভাবে কাজ করবে। এবার তোমার নিজের ক্ষেত্রে একবার পরীক্ষা করে দেখো না! কোন প্যারামিটার ছাড়াই your_age ফাংশনটি call কর। 'Please enter your age' দেখা গেলে কিছু একটা লিখে ফেল:

```
>>> your_age()
Please enter your age
10
you are 10
>>> your_age()
Please enter your age
15
huh?
```

মনে রাখতে হবে টাইপ করার সময় তুমি সংখ্যা হিসেবে ১০ বা ১৫ যা-ই টাইপ কর না কেন, readline সবসময় string রিটার্ন করবে।

পাইথনে অনেক অনেক মডিউল ব্যবহৃত হয়। sys ও time তাদের মধ্যে মাত্র দুটি মডিউল। পাইথনে ব্যবহৃত অন্যান্য মডিউল সম্পর্কে জানতে **Appendix C** দেখ।

৬.৩ এসো নিজে করি:

এই অধ্যায়ে আমরা ফাংশন ও মডিউল ব্যবহার করে পাইথনে একই কোড বারবার ব্যবহারের নিয়ম সম্পর্কে জানলাম। চলক-এর 'scope' (খুব অল্প পরিসরে), ফাংশনের বাইরের চলক, ফাংশনের ভেতরে ব্যবহারের নিয়ম এবং def ব্যবহার করে কীভাবে ফাংশনে তৈরি করা যায় তা শিখলাম।

অনুশীলনী ১

৫ম অধ্যায়ের অনুশীলনী ২ এ আমরা ১০০০০ টাকায় ১০ বছরের সুদ নিয়ে একটি for-loop তৈরি করেছিলাম। এই for-loop কে আমরা সহজেই ফাংশন হিসেবে ব্যবহার করতে পারি। বছরের শুরুতে অর্থের পরিমাণ ও সুদের হারসহ একটি ফাংশন তৈরি করো যার নাম হবে:

```
calculate_interest(100, 0.03)
```

অনুশীলনী ২

সদ্য তৈরি করা ফাংশন থেকে নির্দিষ্ট সময় পর পর সুদ কত হবে তা বের করো- যেমন ৫ বছর বা ১০ বছর। এক্ষেত্রে কোডের নাম হবে:

```
calculate_interest(100, 0.03, 5)
```

অনুশীলনী ৩

প্রদত্ত ফাংশনটিতে প্যারামিটারের সাহায্যে মান পাস (pass) করার পরিবর্তে আমরা একটি ছোট প্রোগ্রাম তৈরি করতে পারি। যেখানে ফাংশন নিজেই মান চাইবে (sys.stdin.readline() ফাংশন ব্যবহার করে)। এক্ষেত্রে কোন প্যারামিটার ছাড়াই আমরা ফাংশনটিকে call করতে পারবো।

```
calculate_interest()
```

এই ছোট প্রোগ্রামটি তৈরিতে আমাদের নতুন একটি ফাংশনের প্রয়োজন হবে যার নাম float। float সম্পর্কে আমরা এখনও আলোচনা করিনি। এটি কিছুটা int ফাংশনের মতো যদিও কিছু ব্যতিক্রম আছে। এটি string-কে ফ্লোটিং পয়েন্ট নম্বরে (floating point numbers) রূপান্তর করে। Floating Point Numbers হলো দশমিক সংখ্যা (.), যেমন- ২০.৩ অথবা ২৫৪১.৯৩৩।

সপ্তম অধ্যায়

ফাইল নিয়ে একটি ছোট অধ্যায়

তোমরা হয়ত ইতোমধ্যে জানো ফাইল কি।

অনেকের বাসায়ই ফাইল ক্যাবিনেট বা বিভিন্ন ফাইল রাখার জন্য অনেক আলমারি দেখা যায়। এখনো বুঝতে পারছো না, আমি কোন জিনিসটির কথা বলছি? ঐ যে লোহার বা স্ট্রিলের একটা বাক্স, যাতে কয়েকটা ড্রয়ার থাকে, আর তার মধ্যে আব্বু-আম্মু তাদের জরুরী কাগজপত্রগুলো (যেগুলো হয়তো আমাদের কাছে আজেবাজে কাগজ মনে হতে পারে) সুন্দর করে এবং যত্ন সহকারে সাজিয়ে রাখে। সেটাই হলো ফাইল ক্যাবিনেট বা ফাইল রাখার আলমারি। সাধারণত এই ফাইলগুলো ফাইলের নামানুসারে বা মাসের নামানুসারে বা বছরের ধারাবাহিকতায় সাজানো থাকে। আবার কখনো কখনো এই ফাইলগুলোর ওপর কাগজে নানা কিছু লেখা থাকে যা দেখে আব্বু-আম্মুরা সহজেই বোঝে সে ফাইলগুলোতে কি লেখা রয়েছে। অর্থাৎ ফাইলের ওপর লেখা সে কথাগুলোই কিন্তু আমাদের অনেক গুরুত্বপূর্ণ তথ্য দেয়। যেমন- ফাইলটি কি, ফাইলটিতে কি ধরনের তথ্য আছে, ফাইলটি কবে খোলা হয়েছে, ফাইলটি কবে শেষ ব্যবহার করা হয়েছে ইত্যাদি। আর ফাইলের আলমারি বা ক্যাবিনেটটি কি করে জানো? এই বিভিন্ন তথ্য সম্বলিত ফাইলগুলোকে গুছিয়ে আলাদা আলাদা করে রাখতে সাহায্য করে। একটু লক্ষ করলে দেখবে আব্বু-আম্মুরা একেকটি ড্রয়ারে একেক ধরনের ফাইল রাখেন। এতে অল্প প্রয়োজনীয়, বেশি প্রয়োজনীয় ফাইলগুলো আলাদা আলাদাভাবে সুন্দর করে সাজিয়ে রাখা সম্ভব হয়। এই একই কাজ আমরা কম্পিউটারে করতে পারি ডিরেক্টরি (directories) বা ফোল্ডারের (folder) মাধ্যমে। কাগজের ফাইলগুলোর উপরের লেখা দেখে যেমন আমরা বুঝতে পারি এই ফাইলে কি আছে, তেমনি কম্পিউটারেও ফাইলের নাম দেখেও আমরা বুঝতে পারি এটা কোন ফাইল। আব্বু-আম্মুর ফাইলগুলোর মধ্যে যেমন জরুরী অনেক তথ্য জমা থাকে, কম্পিউটারের ফাইলগুলোর মধ্যেও সেরকম নানা তথ্য জমা করে রাখতে পারি আমরা।

আগের অধ্যায়ে আমরা দেখেছি পাইথনে কীভাবে ফাইল অবজেক্ট তৈরি করতে হয়, কীভাবে ফাইলে জমা রাখা মূল্যবান সব তথ্য পড়তে হয়:

```
>>> f = open('c:\\test.txt')
>>> print(f.read())
```

f.read() দেখে তোমার মনে হয়ত প্রশ্ন জেগেছে, ফাইল কি শুধুই পড়া যায়? আব্বু-আম্মুর ফাইলগুলো থেকে তথ্য যেমন পড়া যায়, তেমনি নতুন নতুন তথ্য তো যোগ করাও যায়। পাইথনেও কি তা করা যায়? হ্যাঁ, যায় বৈকি। এসো দেখি কেমন করে আমরা ফাইলে আমাদের মনের কথা লিখতে পারি।

প্রথমেই, আমরা যখন পাইথনে ফাইলটি খুলব লেখার জন্য, শুরুতেই তাকে জানিয়ে দেব যে আমরা ফাইলটিতে লিখতে চাই ।

```
>>> f = open('myfile.txt', 'w')
```

আমরা open() ফাংশনের ভিতরে যে অতিরিক্ত "w" টি দিলাম ফাইল এর নামের পরে কমা দিয়ে, এই "w" টা দেখেই পাইথন বুঝতে পারবে ফাইলটি খুলতে হবে লেখার জন্যে । এবার এসো কিছু লেখা যাক:

```
>>> f = open('myfile.txt', 'w')
```

```
>>> f.write('this is a test file')
```

কি করলাম আমরা? একটি ফাইল খুলে তাতে লিখে দিলাম "this is a test file" । আমাদের লেখা শেষ কিন্তু পাইথন এখনো ফাইলটি খুলে বসে আছে, আবার যদি আমরা আরো কিছু লিখতে চাই এই চিন্তা করে। পাইথনকে বরং আমরা বলে দেই যে আমাদের লেখা শেষ, ও যেন ফাইলটি বন্ধ করে দেয়। এই কথাটি বলার জন্য তাহলে আমাদের লিখতে হবে নিচের মতো করে।

```
>>> f = open('myfile.txt', 'w')
>>> f.write('this is a test file')
>>> f.close()
```

f.close() দেখে পাইথন ফাইলটি বন্ধ করে দিবে । আচ্ছা পাইথনকি সত্যিই লিখেছে? এসো পরীক্ষা করে দেখি। কোন টেক্সট এডিটর ব্যবহার করে myfile.txt ফাইলটি খুলে দেখি চলো। অথবা আমরা পাইথনকেই বলতে পারি ফাইলটি আবার পড়তে:

```
>>> f = open('myfile.txt')
>>> print(f.read())
this is a test file
```

কি মজার, তাই না? পাইথন কি সুন্দর করে ফাইল পড়তে ও লিখতে পারে? এবার নিজে নিজে কিছু অনুশীলন কর। দেখবে বিষয়টি কত সহজ হয়ে গেছে।

অষ্টম অধ্যায়

কচ্ছপের মেলা

তোমাদের নিশ্চয়ই মনে আছে ৩য় অধ্যায়ে আমরা একটা "turtle" মডিউল নিয়ে কাজ করেছিলাম।ওখানে ক্যানভাসের উপর ছবি আঁকার জন্য শুরুতে আমাদেরকে মডিউলটি ইম্পোর্ট করে 'Pen' অবজেক্ট তৈরি করে নিতে হয়েছিল। ঠিক এরকম করে:

```
>>> import turtle
>>> t = turtle.Pen()
```

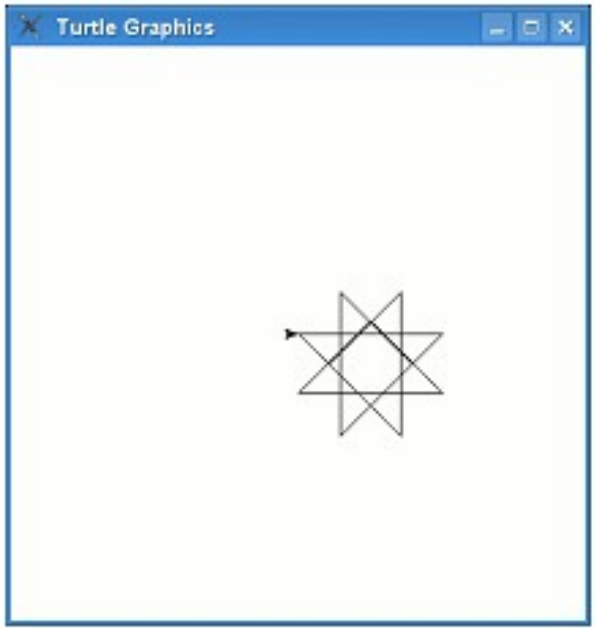
এখন আমরা কিছু সাধারণ ফাংশন ব্যবহার করে ক্যানভাসের ওপর ছোট কচ্ছপটাকে এক স্থান থেকে আরেক স্থানে নিয়ে গিয়ে বিভিন্ন আকার, আকৃতি তৈরি করতে পারি। যেমন আগের অধ্যায়ে আমরা একটা বর্গ এঁকেছিলাম:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

এটাকেই আমরা for-loop ব্যবহার করে নিচের মতো গুছিয়ে লিখতে পারি:

```
>>> t.reset()
>>> for x in range(1,5):
...     t.forward(50)
...     t.left(90)
...
```

এভাবে লিখতে অনেক কম সময় লাগল। এবার চলো আরো মজার কিছু দেখা যাক:



চিত্র ৮.১: কচ্ছপটি কি সুন্দর ৮ মাথাওয়ালা একটি তারা এঁকে ফেলল

```
>>> t.reset()
>>> for x in range(1,9):
...     t.forward(100)
...     t.left(225)
...
```

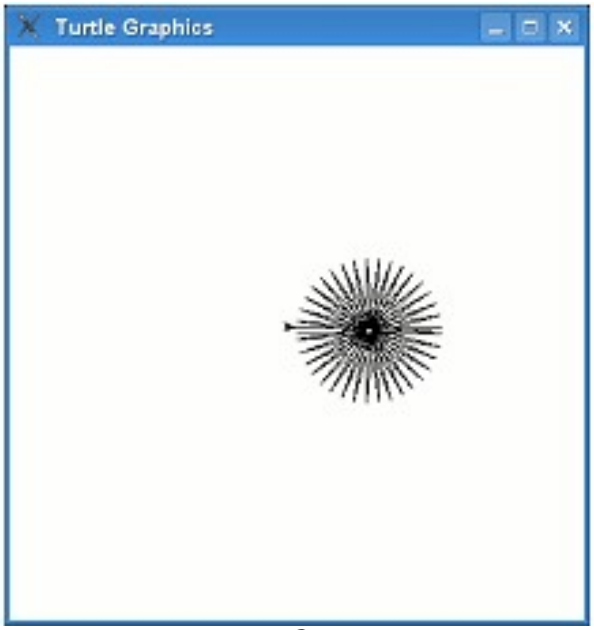
উপরের এই কোড ব্যবহার করেই অষ্ট কৌণিক এই তারাটি আঁকা হল। চিত্র ৮.১ এ তারাটি দেখান হল (কচ্ছপটি প্রতিবার ১০০ পিক্সেল সামনে এগিয়ে ২২৫ ডিগ্রী মোড় নেয়)।

আমরা চাইলে কোণের পরিবর্তন (১৭৫ ডিগ্রী) এবং দীর্ঘ লুপ (৩৭বার) ব্যবহার করে আরো বেশী কোণ সংযুক্ত তারা তৈরি করতে পারি:

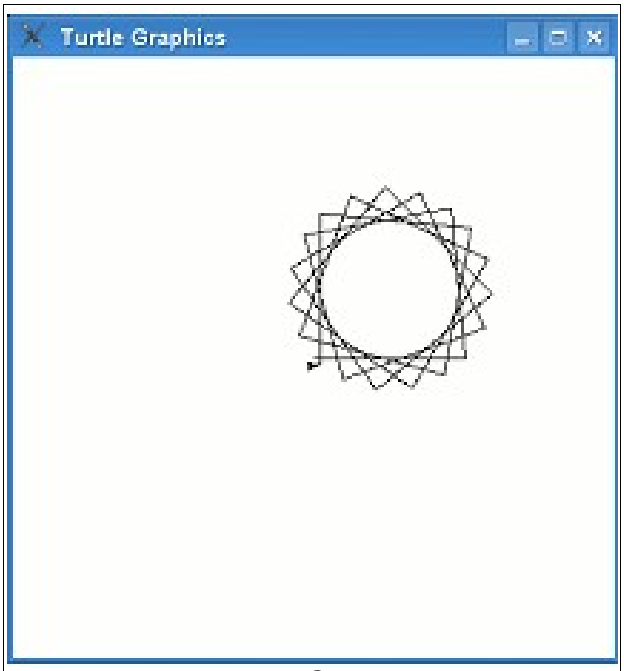
```
>>> t.reset()
>>> for x in range(1,38):
...     t.forward(100)
...     t.left(175)
...
```

অথবা পেঁচানো তারাও তৈরি করতে পারি:

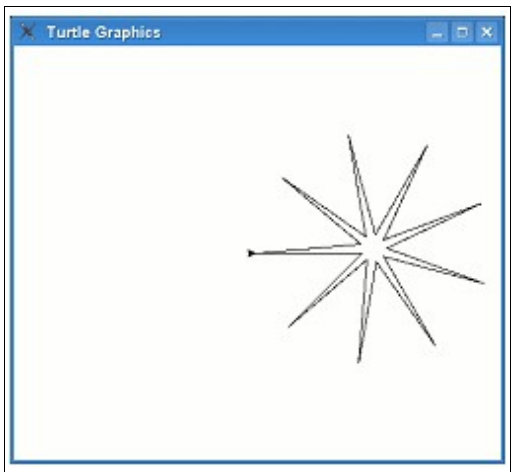
```
>>> for x in range(1,20):
...     t.forward(100)
...     t.left(95)
...
```



চিত্র ৮.২: আরো বেশী মাথাওয়ালা তারা



চিত্র ৮.৩: আরো বেশী মাথাওয়ালা তারা



চিত্র ৮.৪: নয় মাথাওয়ালা তারা

এবার চল আরেকটু জটিল কিছু করি:

```
>>> t.reset()
>>> for x in range(1,19):
...     t.forward(100)
...     if x % 2 == 0:
...         t.left(175)
...     else:
...         t.left(225)
...
```

উপরের কোডে আমরা পরীক্ষা করে দেখছি x জোড় সংখ্যা কিনা - এর জন্য আমরা মডুল (modulo) অপারেটর (%) ব্যবহার করেছি - if x % 2 == 0 ।

x % 2 এর মান শূন্য হবে যদি x ২ দ্বারা বিভাজ্য হবে । যদি বুঝতে সমস্যা হয় তাহলে এগুলো নিয়ে খুব দুশ্চিন্তা করো না। শুধু মনে রেখ "x % 2 == 0" ব্যবহার করে আমরা জোড় সংখ্যা পরীক্ষা করি। এই কোড রান করলে আমরা নয় মাথাওয়ালা একটি তারা পাব।

শুধু তারা কিংবা সাধারণ জ্যামিতিক আকার নয়, কচ্ছপটি বিভিন্ন ফাংশনের সংমিশ্রণে অনেক কিছু আঁকতে পারে। যেমন:

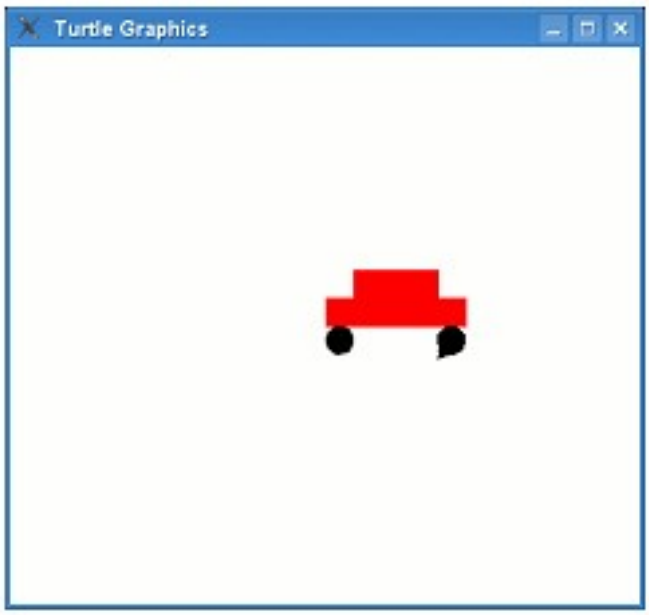
```
t.color(1,0,0)
t.begin_fill()
t.forward(100)
t.left(90)
t.forward(20)
t.left(90)
t.forward(20)
t.right(90)
t.forward(20)
t.left(90)
t.forward(60)
t.left(90)
```

```
t.forward(20)
t.right(90)
t.forward(20)
t.left(90)
t.forward(20)
t.end_fill()
t.color(0,0,0)
t.up()
t.forward(10)
t.down()
t.begin_fill()
t.circle(10)
t.end_fill()
t.setheading(0)
t.up()
t.forward(90)
t.right(90)
t.forward(10)
t.setheading(0)
t.begin_fill()
t.down()
t.circle(10)
t.end_fill()
```

অনেক সময় নিয়ে, অনেক কোড লিখে আমরা পেলাম একটি পুরানো আমলের গাড়ি। চিত্র: ৮.৫-এ আমরা তার ছবি দেখতে পাবো। কিন্তু এ থেকে আমরা নতুন কিছু জিনিস শিখলাম। যেমন: রং এর ব্যবহার, কীভাবে কলমের কালি পরিবর্তন করে নিতে হয়, কীভাবে ক্যানভাসের একটি নির্দিষ্ট এলাকা রং দিয়ে পূর্ণ করতে হয়, কীভাবে নির্দিষ্ট মাপের একটি বৃত্ত আঁকতে হয়।

৮.১ রং করা

color ফাংশনটি ৩টি প্যারামিটার গ্রহণ করে। প্রথমটি লাল, দ্বিতীয়টি সবুজ এবং শেষের প্যারামিটারটি নীল রং-এর পরিমান নির্দেশ করে। এখানে মূলত আমরা আরজিবি (RGB) পরিমাপের মাধ্যমে একটি নির্দিষ্ট রং ফুটিয়ে তুলছি।



চিত্র ৮.৫: কচ্ছপ দিয়ে গাড়ি আঁকা আসলেই কঠিন কাজ!!!

লাল, সবুজ এবং নীল কেন?

যদি তুমি কখনো বিভিন্ন রং নিয়ে খেলা করে থাক তাহলে তুমি এই প্রশ্নের উত্তর ইতোমধ্যে জেনে গেছ। যখন তুমি দুটি রং-এর মিশ্রণ ঘটাও তখন অন্য আরেকটি রং পাও। যখন লাল আর নীল মেশাও তখন পাও বেগুনী, আর যখন অনেকগুলো রং একসাথে মেশাও তখন পাও কাদার মত খয়েরী রং। কম্পিউটারেও তুমি একইভাবে রং মেশাতে পার – লাল আর সবুজ মেশাও হলুদ পাবে - শুধু ব্যতিক্রম কম্পিউটারে আমরা আলোর রং মেশাই, তরল রং নয়।

যদিও আমরা আসল রং ব্যবহার করছি না, চিন্তা কর তিনটি বড় পাত্রে রং আছে। একটায় লাল, একটায় সবুজ, একটায় নীল। প্রত্যেকটি পাত্রই পূর্ণ, আমরা ধরে নিব সম্পূর্ণ পাত্রের মান হল ১ (বা ১০০%)। আমরা একটা পাত্রে সব টুকু লাল রং (১০০%) ঢেলে দেই, এবার তাতে সবটুকু সবুজ (১০০%) মেশাই। কিছুক্ষণ পর রং দুটো মিশে হলুদ হয়ে যাবে। এবার তবে এসো একটা হলুদ বৃত্ত তৈরি করি কচ্ছপটাকে দিয়ে।

```
>>> t.color(1,1,0)
>>> t.begin_fill()
>>> t.circle(50)
>>> t.end_fill()
```

উপরের উদাহরণে আমরা color ফাংশনে ১০০% লাল, ১০০% সবুজ আর ০% নীল ব্যবহার করেছি। সহজে রং নিয়ে পরীক্ষা-নিরীক্ষা করার জন্য আমরা এটাকে একটা ফাংশনে রূপান্তরিত করতে পারি।

```
>>> def mycircle(red, green, blue):
...             t.color(red, green, blue)
...       t.begin_fill()
...       t.circle(50)
...       t.end_fill()
...
```

আমরা একটি উজ্জ্বল সবুজ বৃত্ত আঁকতে পারি সবটুকু সবুজ ব্যবহার করে (১০০%):

```
>>> mycircle(0, 1, 0)
```

এবং একটি গাঢ় সবুজ বৃত্ত আঁকতে পারি অর্ধেক সবুজ রং (৫০%) দিয়ে :

```
>>> mycircle(0, 0.5, 0)
```

এখানে আসলে রং নিয়ে ভাবলে খুব একটা বোঝা যাবে না। বাস্তবে এক পাত্র রং থাকলে তুমি যতটুকুই ব্যবহার কর না কেন একই রকম দেখাবে। কম্পিউটারে আমরা আলো নিয়ে খেলা করছি, তাই কোন রং একটু কম ব্যবহার করলে সেটা গাঢ় দেখাবে। রাত্রে টর্চ লাইট জ্বালালে যেমন প্রথমে উজ্জ্বল হলদে আলো পাওয়া যায়, ব্যাটারি শেষ হতে থাকলে তা আসতে আসতে ফিকে হয়ে আসে, হলুদাভ বর্ণটি আসতে আসতে গাঢ় হয়ে আসে এখানেও তেমনি ঘটে। নিজেই চেষ্টা করে দেখ না একটা পূর্ণ লাল আর অর্ধ লাল এবং পূর্ণ নীল আর অর্ধ নীল বৃত্ত এঁকে কী অবস্থা হয়:

```
>>> mycircle(1, 0, 0)
>>> mycircle(0.5, 0, 0)
>>> mycircle(0, 0, 1)
>>> mycircle(0, 0, 0.5)
```

নানা রং-এর সংমিশ্রণে বৈচিত্র্যময় অনেক রং পাওয়া যাবে। সোনালী রং পাওয়া যেতে পারে ১০০% লাল, ৮৫% সবুজ এবং কোন নীল ব্যবহার না করে :

```
>>> mycircle(1, 0.85, 0)
```

হালকা গোলাপী পাওয়া যাবে ১০০% লাল, ৭০% সবুজ আর ৭৫% নীল ব্যবহার করে:

```
>>> mycircle(1, 0.70,0.75)
```

কমলা রং পাবে ১০০% লাল, ৬৫% সবুজ মিশিয়ে আর খয়েরী রং পাওয়া যাবে ৬০% লাল, ৩০% সবুজ আর ১৫% নীল মিশিয়ে:

```
>>> mycircle(1, 0.65, 0)
>>> mycircle(0.6, 0.3, 0.15)
```

যে কোন সময়ে ক্যানভাস পরিষ্কার করতে চাইলে আমাদের ব্যবহার করতে হবে t.clear()

৮.২ অন্ধকার

আচ্ছা, যখন আমরা রাতের বেলায় আলো নিভিয়ে দেই তখন কি হয়? সব কিছু অন্ধকার হয়ে যায়। রং এর বেলায় ও ঠিক একই ঘটনা ঘটে। যখন কোন বর্ণ থাকে না তখন সেটা কালো দেখায়।

সুতরাং আমরা যদি একটি বৃত্তের ক্ষেত্রে লাল, নীল এবং সবুজ – তিনটি বর্ণের মানই 0 দেই তাহলে সেটিও একটি কাল বর্ণের বৃত্ত তৈরি করবে, চিত্র ৮.৬-এ আমরা তা দেখতে পাবো।

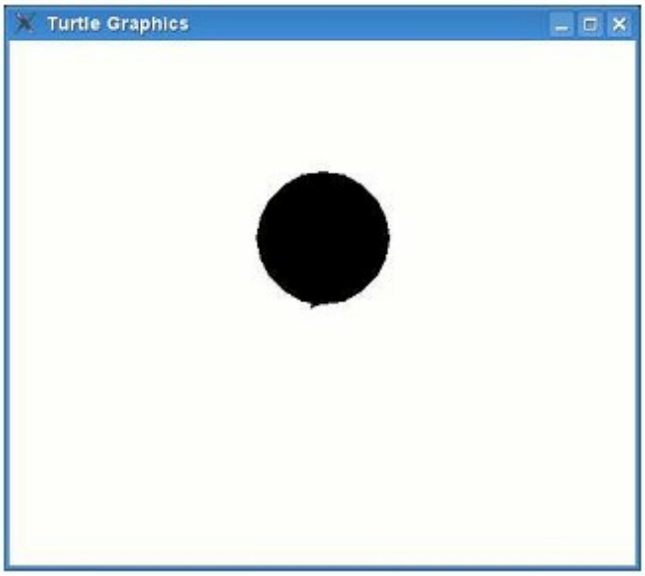
```
>>> mycircle(0, 0, 0)
```

আবার এর উল্টোটাও সত্যি। যদি সবগুলোর মান ১০০% করে দেই তাহলে আমরা সাদা বর্ণ পাবো।

```
>>> mycircle(1,1,1)
```

৮.৩ রং দিয়ে ভরাট করা

তুমি হয়ত এতক্ষণে বুঝতে পেরেছ যে fill ফাংশনটি সক্রিয় করতে হয় প্যারামিটার হিসেবে - 1 প্রদান বা পাস করে। আবার নিষ্ক্রিয় করতে 0 ব্যবহার করা হয়। যখন আমরা এটাকে নিষ্ক্রিয় করি তখন এটা আমরা যে এলাকাটিতে এঁকেছি সেটাকে রং দিয়ে ভরাট করে দেয়। তবে এজন্য অবশ্যই ফাংশনটিকে সক্রিয় করার পর কোন একটা আকৃতির নূণ্যতম একটা অংশ আঁকতে হবে। সুতরাং এভাবে আমরা আমাদের আগের লেখা কোড ব্যবহার করে খুব সহজেই একটি বর্গক্ষেত্রকে রং দিয়ে পূর্ণ করতে পারি। তবে প্রথমে বর্গক্ষেত্র আঁকার বিষয়টিকে একটি ফাংশনে রূপান্তরিত করা যাক। বর্গক্ষেত্র আঁকতে আমরা এই কোড লিখি:



চিত্র ৮.৬: একটি কালো বৃত্ত

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

ফাংশন হিসেবে আমরা চাইতে পারি বর্গক্ষেত্রের আকারকেই আমরা প্যারামিটার হিসেবে পাস করব। এতে ফাংশনটি ব্যবহার করা আরো সহজ হবে:

```
>>> def mysquare(size):
...         t.forward(size)
...         t.left(90)
```

```
...         t.forward(size)
...         t.left(90)
...         t.forward(size)
...         t.left(90)
...         t.forward(size)
...         t.left(90)
```

এবার ফাংশনটিকে আমরা পরীক্ষা করে দেখতে পারি এভাবে:

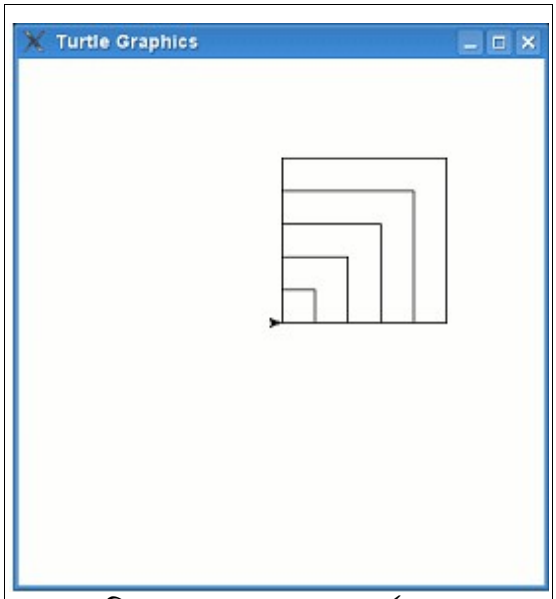
```
>>> mysquare(50)
```

আমরা যদি উপরের কোড পর্যবেক্ষণ করি তবে দেখব আমরা forward(size) এবং left(90) চারবার পুনরাবৃত্তি করেছি। এত কষ্ট না করে আমরা একটা for-loop ব্যবহার করে সহজেই কাজটি করতে পারি এভাবে:

```
>>> def mysquare(size):
...         for x in range(0,4):
...             t.forward(size)
...             t.left(90)
```

আগের সংস্করণের তুলনায় এটিতে বেশ বড় মাপের উন্নয়ন লক্ষ করা যায়। এবার বিভিন্ন আকার নিয়ে চেষ্টা করা যাক:

```
>>> t.reset()
>>> mysquare(25)
>>> mysquare(50)
>>> mysquare(75)
>>> mysquare(100)
>>> mysquare(125)
```



চিত্র ৮.৭: অনেকগুলো বর্গক্ষেত্র

উপরের চিত্রের মত আকৃতি তৈরি হবে।

এখন আমরা একটা রং দিয়ে পূর্ণ বর্গ তৈরি করার চেষ্টা করি। প্রথমেই ক্যানভাসটিকে পরিষ্কার করে নেই:

```
>>> t.reset()
```

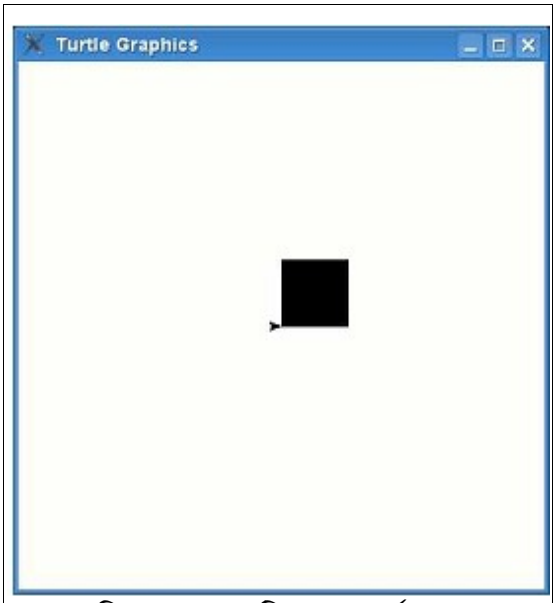
এবার fill ফাংশনকে সক্রিয় করে বর্গ আঁকি:

```
>>> t.begin_fill()
>>> mysquare(50)
```

fill ফাংশনকে নিষ্ক্রিয় না করা পর্যন্ত শুধু একটা বর্গ দেখা যাবে।

```
>>> t.end_fill()
```

এবার নিচের ছবির মত একটি বর্গ দেখা যাবে:



চিত্র ৮.৮: একটি কালো বর্গক্ষেত্র

এবার আমরা ফাংশনটিকে এমনভাবে পরিবর্তন করব যেন এটি ব্যবহার করে আমরা রং দ্বারা পূর্ণ এবং রংহীন বর্গক্ষেত্র আঁকতে পারি। এজন্য আমাদেরকে একটু জটিল কোড লিখতে হবে:

```
>>> def mysquare(size, filled):
...             if filled == True:
...                     t.begin_fill()
...             for x in range(0,4):
...                     t.forward(size)
...                     t.left(90)
...             if filled == True:
...                     t.end_fill()
...
```

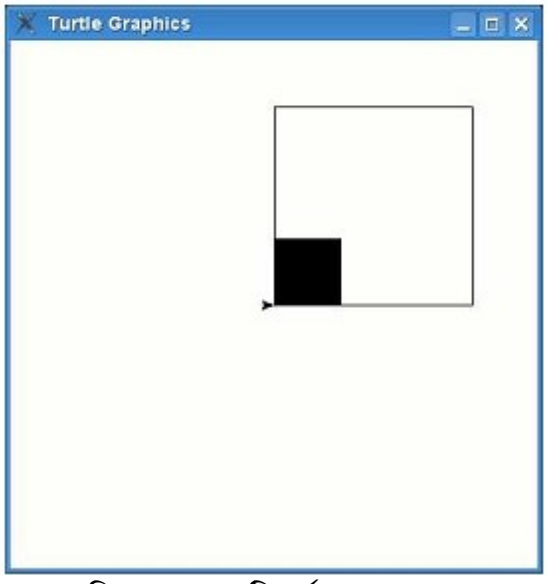
এখন চলো প্রথম দুই লাইন পরীক্ষা করে দেখি filled প্যারামিটারের মান True কিনা। যদি True হয় তাহলে fill

ফাংশনকে সক্রিয় করি। চারবার লুপের মাধ্যমে বর্গটি আঁকা শেষে আবার পরীক্ষা করে দেখি filled প্যারামিটারের মান True কিনা। যদি True হয় তবে fill ফাংশনকে নিষ্ক্রিয় করে দেয়। এবার একটা রং দিয়ে ভরাটকৃত বর্গ আঁকতে পারি এভাবে:

```
>>> mysquare(50, True)
```

আর রং ছাড়া বর্গ আঁকার জন্য:

```
>>> mysquare(150, False)
```



চিত্র ৮.৯: একটি বর্গক্ষেত্রের চোখ

উপরের কোড চালালে এমন একটি আকৃতি পাব আমরা যেটা দেখতে অনেকটা বর্গাকৃতির চোখের মত।
এভাবে তুমি সব ধরনের আকার আকৃতি তৈরি করতে পার এবং সেগুলোকে রং দিয়ে ভরাট করে দিতে পার। আমরা আগে যে তারাটি এঁকেছিলাম এসো সেটিকে একটি ফাংশনে পরিণত করি। এটা ছিল আমাদের আসল কোড:

```
>>> for x in range(1,19):
...         t.forward(100)
...         if x % 2 == 0:
...             t.left(175)
...         else:
...             t.left(225)
...
```

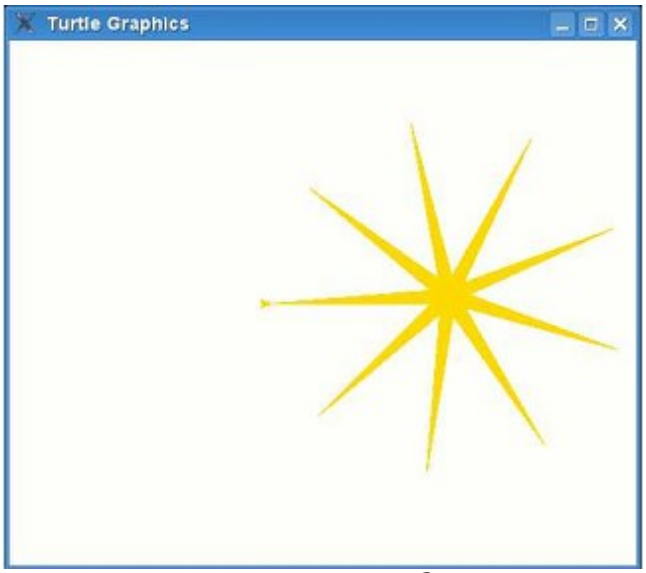
আমরা mysquare ফাংশনের মত একই if-statements ব্যবহার করতে পারি। size প্যারামিটারটিকে আমরা forward ফাংশনে পাস করতে পারি।

```
1. >>> def mystar(size, filled):
```

```
2. ...     if filled:
3. ...         t.begin_fill()
4. ...     for x in range(1,19):
5. ...         t.forward(size)
6. ...         if x % 2 == 0:
7. ...             t.left(175)
8. ...         else:
9. ...             t.left(225)
10. ...    if filled:
11. ...        t.end_fill()
```

এখন filled প্যারামিটারের মানের ওপর ভিত্তি করে ২য় এবং ৩য় লাইনে আমরা রং দিয়ে ভরাট করা শুরু করি। লাইন ১০ এবং ১১ তে এসে ঠিক তার উল্টোটা করি। size প্যারামিটারের মান আমরা ৫ম লাইনে ব্যবহার করি। এবার আমরা রং হিসেবে সোনালী বর্ণ ব্যবহার করব। নিশ্চয়ই তোমাদের মনে আছে সোনালী বর্ণ বানাতে ১০০% লাল, ৮৫% সবুজ এবং ০% নীল লাগে। এবার আমরা ফাংশনটিকে কল করব:

```
>>> t.color(1, 0.85, 0)
>>> mystar(120, True)
```



চিত্র ৮.১০: একটি সোনালী তারা

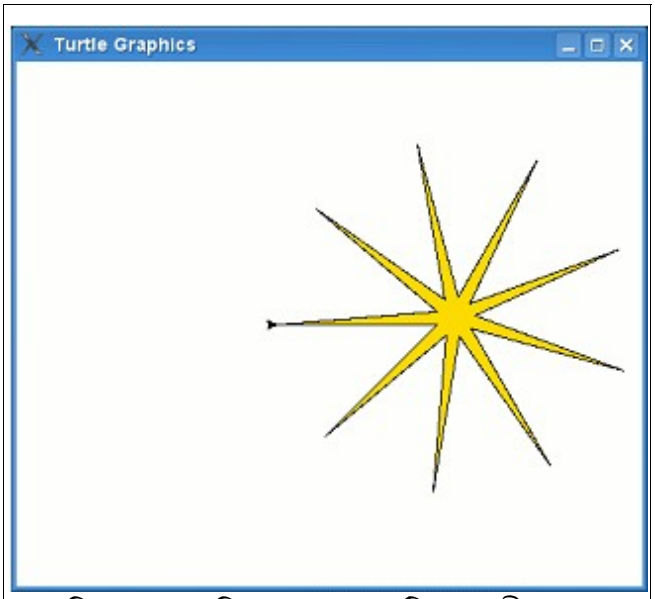
উপরের ছবির মত একটা স্বর্ণালী তারা তৈরি হবে। এবার আমরা তারার বর্ণ পরিবর্তন করে একটা আউটলাইন তৈরি করতে পারি:

```
>>> t.color(0,0,0)
>>> mystar(120, False)
```

এবার আমরা রং হিসেবে কাল এবং ফিলিং নিষ্ক্রিয় করে আরেকটি তারা এঁকেছি। এবার ক্যানভাসটা দেখতে নিচের মত হবে।

৮.৪ নিজে করি

এই অধ্যায়ে আমরা কচ্ছপ মডিউল সম্বন্ধে শিখেছি এবং তা ব্যবহার করে কিছু সাধারণ জ্যামিতিক আকার আঁকাতে শিখেছি। আমরা ফাংশন ব্যবহার করা শিখেছি আমাদের করা কিছু কোড পুনরায় ব্যবহার করার জন্য। এর ফলে সহজেই আমরা বিভিন্ন রং দিয়ে বিভিন্ন আকৃতি আঁকতে পেরেছি।



চিত্র ৮.১১: বহিঃরেখা সহ একটি সোনালী তারা

অনুশীলন – ১:

আমরা বর্গ এঁকেছি, একটা অক্টাগন আঁকলে কেমন হয়? অক্টাগনের ৮টি মাথা থাকে।

(সূত্র: ৪৫ ডিগ্রী কোণে ঘুরিয়ে চেষ্টা করতে হবে।)

অনুশীলন – ২:

অক্টাগন আকার কোডটিকে ফাংশনে রূপান্তরিত করতে হবে এবং এটাকে যে কোন বর্ণ বা রং দ্বারা ভরাট করতে হবে।

নবম অধ্যায়

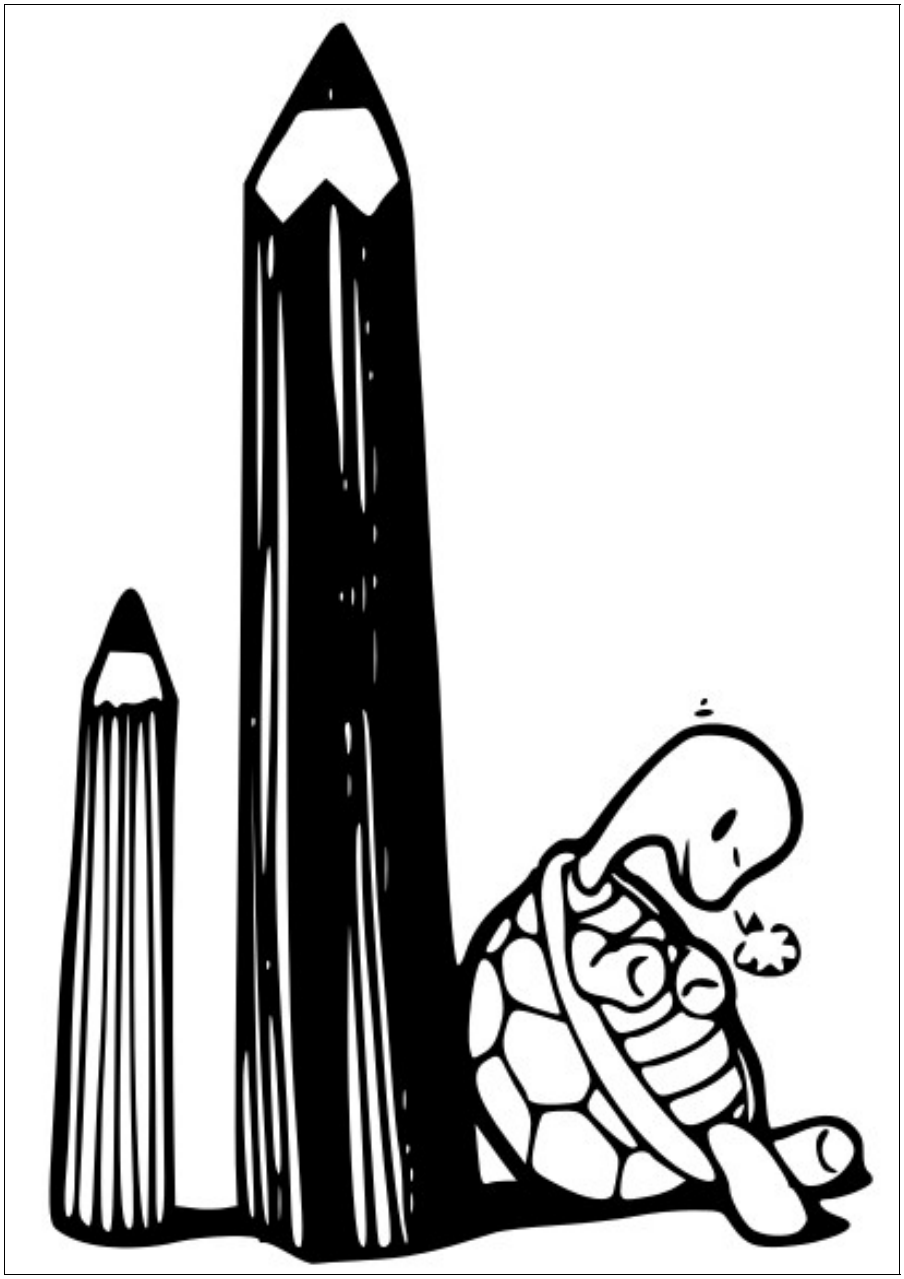
একটুখানি গ্রাফিক্স

কাছিম ব্যবহার করে ছবি আঁকার সমস্যাটি হল.....কাছিম....খুবই.....ধীর.....গতির!!!

যদি একটি কাছিম তার সর্বোচ্চ গতিতেও এগিয়ে যায়, তখনও এটি খুব ধীর গতির। আসলে কাছিমদের জন্য এটি কোন ব্যাপার না বুঝলে? তাদের নষ্ট করার মত প্রচুর সময় রয়েছে। কিন্তু যখন তুমি কম্পিউটার গ্রাফিক্স নিয়ে কাজ করবে তখন এটি একটি বড় ধরনের সমস্যা। যদি তোমার নিন্টেন্ডো, গেমবয় এডভান্স অথবা কম্পিউটারে কোন গেম খেলে থাক, তাহলে এক মুহুর্তের জন্য সেই গেমের গ্রাফিক্স নিয়ে চিন্তা কর (যা তোমরা স্ক্রীনে দেখতে পাও)। গেমের ভেতর গ্রাফিক্স প্রদর্শন করার বিভিন্ন উপায় রয়েছে: একটি হল, টু-ডি (অর্থাৎ দ্বিমাত্রিক) গেম, যেখানে গেমের চিত্রসমূহ সমতল হয় এবং গেমের চরিত্রগুলো শুধু ডান-বাম এবং উপর-নিচ এভাবে নড়াচড়া করতে পারে। নিন্টেন্ডো, গেমবয় অথবা মোবাইল ফোনে এটা আমরা সচরাচর দেখতে পাই। আরও রয়েছে সুডো-থ্রীডি (প্রায় ত্রিমাত্রিক) গেম, যেখানে গেমের চিত্র কিছুটা বাস্তবসম্মত হয়ে থাকে, কিন্তু এখানেও গেমের চরিত্রগুলো ডান-বাম উপর-নিচ ছাড়া আর কোন দিকে নড়াচড়া করতে পারে না। আবারও বলতে হয় বহনযোগ্য গেমিং ডিভাইস অথবা মোবাইল ফোনে আমরা এধরনের গেম দেখতে পাই। অবশেষে রয়েছে থ্রীডি (ত্রিমাত্রিক) গেম। যেখানে স্ক্রীনে প্রদর্শিত চিত্রে বাস্তবতার অনুকরণের প্রচেষ্টা করা হয়।

এই সকল গ্রাফিক্স প্রদর্শন করতে সাধারণভাবে একটি জিনিস প্রয়োজন হয়, সেটি হল স্ক্রীনে খুব দ্রুততার সাথে অঙ্কন করা। তুমি কি কখনও অ্যানিমেশন করার চেষ্টা করেছ? যেখানে তোমার একটি পেন্সিল ও প্যাড থাকবে এবং তুমি প্রথম পৃষ্ঠার এক কোণায় কিছু অঙ্কন করবে (হতে পারে একটি মানুষের ছবি)। পরবর্তী পৃষ্ঠার কোণায় তুমি একই ছবি আঁকবে তবে এবার মানুষটির পা কিছুটা সরে আসবে। পরবর্তী পৃষ্ঠায় তুমি একই ছবি আঁকবে যেখানে মানুষটির পা আরও সরে আসবে। এইভাবে তুমি আরও কিছু পৃষ্ঠাতে ছবিটি আঁকলে। যখন শেষ হবে তখন তুমি পৃষ্ঠাগুলোর পাতা দ্রুত উল্টাবে, যদি খুব দ্রুততার সাথে তুমি পৃষ্ঠা উল্টাও তাহলে মনে হবে মানুষটি হাঁটছে। গেমসে অথবা টিভিতে প্রদর্শিত সকল অ্যানিমেশনের এটিই হল মৌলিক উপায়। তুমি কিছু একটা আঁকলে তারপর তুমি আবারও আঁকলে কিন্তু কিছুটা পরিবর্তিত রূপে এভাবেই দর্শকের চোখে বিভ্রান্তি সৃষ্টি করা হয়। একারণেই কাছিম সকল গ্রাফিক্সের কাজের জন্য উত্তম নয়। যদি তুমি এরকম ছবি আঁকতে চাও যা দেখে মনে হবে যে ছবিটি নড়াচড়া করছে তাহলে তোমাকে প্রতিটি ফ্রেমের ছবি খুব দ্রুততার সাথে অঙ্কন করতে হবে। ঘাবড়ে যেও না, এ কাজটি করবে তোমার কম্পিউটার।

ত্রিমাত্রিক গ্রাফিক্সের চেয়ে দ্বিমাত্রিক গ্রাফিক্স যথেষ্ট আলাদাভাবে তৈরি করা হয়ে থাকে, কিন্তু তারপরও মৌলিক উপায়টি একই থাকে। যখন তোমার কাছিমটি একটি অঙ্কনের ক্ষুদ্র অংশ এঁকে শেষ করবে তখনই পরবর্তী অঙ্কনের জন্য পাতা উল্টাতে হবে...



৯.১ দ্রুত অঙ্কন

স্ক্রীনে অঙ্কন করার জন্য ভিন্ন ভিন্ন প্রোগ্রামিং ভাষা ভিন্ন ভিন্ন পদ্ধতি অবলম্বন করে থাকে। কিছু পদ্ধতি দ্রুত গতির আবার কিছু আছে ধীর গতির, এর মানে হচ্ছে যে সকল প্রোগ্রামার নিজেদের জীবনধারণের জন্য গেম তৈরি করে থাকে তাদেরকে প্রোগ্রামিং ভাষা নির্বাচন করার ব্যাপারে বিশেষভাবে সতর্ক থাকতে হবে।

পাইথনেরও গ্রাফিক্সের কাজের জন্য কিছু ভিন্ন উপায় রয়েছে (যার মধ্যে রয়েছে পূর্বে আলোচনা করা কাছিম দ্বারা অঙ্কনের পদ্ধতি), কিন্তু সাধারণত গ্রাফিক্স-এর জন্য সেরা পদ্ধতি হচ্ছে মডিউল এবং কোডের লাইব্রেরী যা পাইথনের সাথে সমন্বিত থাকে না। তোমাকে সম্ভবত এই ধরণের জটিল লাইব্রেরী ইনস্টল এবং ব্যবহার করার আগে কয়েক বছর প্রোগ্রামিং করতে হবে।

সৌভাগ্যবশত পাইথনের সাথেই একটি মডিউল রয়েছে, যার সাহায্যে আমরা মৌলিক গ্রাফিক্সের কাজ করতে পারব (কাছিমের চাইতে কিছুটা দ্রুতগতি সম্পন্ন)। সম্ভবত এতটাই দ্রুত যে আমরা এর নাম দিতে পারি Quick Draw Turtle বা দ্রুত অঙ্কনের কাছিম।

মডিউলটির নাম হল tkinter (একটা অদ্ভুত নাম তাই না ? এর মানে হচ্ছে 'Tk interface' বা 'টি কে ইন্টারফেস') এবং এর সাহায্যে একটি পরিপূর্ণ অ্যাপ্লিকেশন তৈরি করা সম্ভব (হ্যাঁ, এমনকি তুমি চাইলে এর সাহায্যে একটি সাধারণ ওয়ার্ড প্রসেসর প্রোগ্রাম তৈরি করে ফেলতে পারবে) এবং সেইসাথে সাধারণ অঙ্কনের কাজও করা যায়। আমরা বাটনের সাথে একটি সাধারণ অ্যাপ্লিকেশন তৈরি করতে পারি নিম্নে উল্লেখ করা কোড অনুসারে:

```
>>> from tkinter import*
1. >>> tk=Tk()
2. >>> btn = Button(tk,text="click me")
3. >>> btn.pack()
```

প্রথম লাইনে, আমরা Tk মডিউলের বিষয়বস্তু ইমপোর্ট করেছি, যাতে আমরা তা ব্যবহার করতে পারি-এগুলোর মধ্যে সবচেয়ে দরকারী হচ্ছে Tk, যা একটি মৌলিক উইন্ডো তৈরি করে এবং পরবর্তীতে আমরা এর মধ্যে অনেক কিছু সংযোজন করতে পারি। তুমি দ্বিতীয় লাইনটি লিখলে, এই উইন্ডোটি স্ক্রীনে প্রদর্শিত হবে। তৃতীয় লাইনে, আমরা একটি নতুন বাটন তৈরি করব এবং btn নামক ভ্যারিয়েবলে তা অ্যাসাইন করব। এই লাইনে tk অবজেক্ট একটি প্যারামিটার হিসেবে রয়েছে এবং নামের প্যারামিটার হিসেবে রয়েছে দুটো শব্দ 'click me'.

নামের প্যারামিটার

নামের প্যারামিটার এই প্রথম আমরা ব্যবহার করছি। এই জিনিসটি সাধারণ প্যারামিটারের মতই কাজ করে, ব্যাতিক্রম হচ্ছে এটি যেকোনোভাবে সন্নিবিষ্ট হতে পারে, তাই আমাদেরকে এর জন্য একটি নাম নির্বাচন করতে হবে।

উদাহরণস্বরূপ ধরে নাও আমাদের rectangle নামক একটি ফাংশন রয়েছে যার দুটো প্যারামিটার রয়েছে width অর্থাৎ প্রস্থ এবং height অর্থাৎ উচ্চতা। সাধারণত আমরা যা করি তা হল rectangle(200, 100) এভাবে ফাংশন কল করি। যার মানে হচ্ছে rectangle এর প্রস্থ ২০০ পিক্সেল এবং উচ্চতা ১০০ পিক্সেল। কিন্তু কি হবে যদি প্যারামিটার দুটো উল্টাপাল্টা হয়ে যায়? অর্থাৎ প্যারামিটার যেকোনো ক্রম বজায় রাখলে কি হবে? সব কিছু ওলটপালট হয়ে যাবে তাই না? তাহলে আমাদের কি করা উচিৎ? কোনটি উচ্চতা এবং কোনটি প্রস্থ তা বলে দিতে হবে। যেমন: rectangle(height=100, width=200) এবার উল্টাপাল্টা হবার কোন সম্ভাবনাই নেই। আসলে নামের প্যারামিটার ব্যবহারের কারণটি জটিল, এটি ফাংশনকে সহজভাবে কাজ করার জন্য ব্যবহার করা হয়-কিন্তু এই বিষয় সম্পর্কে আরও উচ্চতর পর্যায়ের বইতে আলোচনা করা হয়ে থাকে।

সর্বশেষ লাইন অর্থাৎ ৪ নাম্বার লাইনে বাটনকে নির্দেশ দেয়া হয় নিজে নিজে অঙ্কন করার জন্য। ২ নাম্বার লাইনে যে উইন্ডো তৈরি হয়েছিল তা একটি ছোট্ট বাটনের আকৃতি ধারণ করবে এবং এতে লিখা থাকবে 'click me'. এটা দেখতে অনেকটা এরকম হবে:

বাটনটি আসলে কোন কাজই করে না, তুমি শুধু এতে ক্লিক করতে পারবে কিন্তু আর কিছু হবে না। আমরা বাটনকে দিয়ে কাজ করাতে পারব পূর্বের উদাহরণটি কিছুটা পরিবর্তন করে (তার আগে নিশ্চিত হয়ে নাও যে পূর্বের করা উইন্ডোটি তুমি বন্ধ করেছ।)। প্রথমে আমরা একটি ফাংশন তৈরি করতে পারি কিছু লেখা পয়েন্ট করে রাখার জন্য:

```
>>> def hello():
…    print('hello there')
```

এরপর পূর্বের কোডটি পরিবর্তন করব যাতে এই ফাংশনটি ব্যবহার করতে পারে:

```
>>> from tkinter import*
>>> tk=Tk()
>>> btn=Button(tk,text="click me",command=hello)
>>> btn.pack()
```

নামের প্যারামিটার 'command' বলে দিচ্ছে যে আমরা hello ফাংশনটি ব্যবহার করতে চাই, যখন বাটনে ক্লিক করা

হবে। এখন যদি তুমি বাটনে ক্লিক কর তাহলে দেখবে 'hello there' লেখাটি আসছে। যতবার তুমি বাটনে ক্লিক করবে ততবার তুমি এটি দেখতে পাবে।

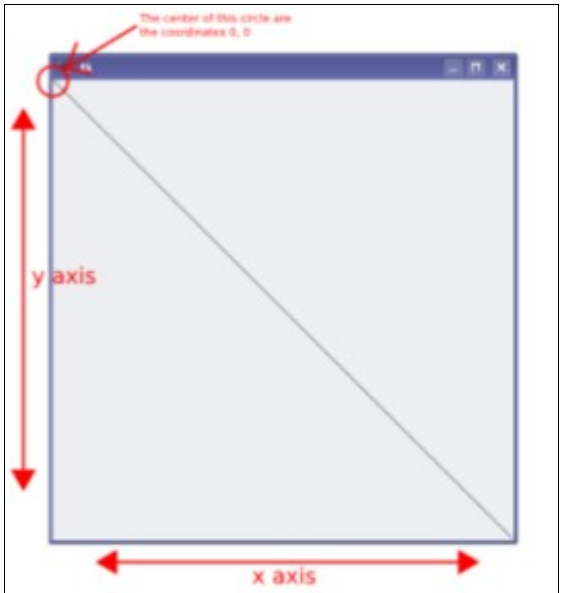
৯.২ সাধারণ অঙ্কন

স্ক্রীনে অঙ্কন করার জন্য বাটন খুব বেশি সহায়ক নয়-তাই আমাদের নতুন একটি উপাদান তৈরি করে সংযুক্ত করতে হবে: একটি ক্যানভাস। ক্যানভাস তৈরি করার সময়, আমাদের প্রস্থ এবং উচ্চতা বলে দিতে হবে, কিন্তু বাটন তৈরির সময় আমরা এটি না করে শুধু টেক্সট এবং কমান্ড প্যারামিটার বলে দিয়েছিলাম। এই পার্থক্যটুকু ছাড়া বাকী সবকিছুই আগের কোডের মত:

```
>>> from tkinter import *
>>> tk=Tk()
>>> canvas=Canvas(tk,width=500,height=500)
>>> canvas.pack()
```

বাটনের উদাহরণের মত, দ্বিতীয় লাইন টাইপ করলে একটি উইন্ডো প্রদর্শিত হবে। যখন চার নাম্বার লাইনে আমরা ক্যানভাসটি প্যাক (pack) করব হঠাৎ করেই এর আকৃতি বৃদ্ধি পাবে। পিক্সেল স্থানাঙ্ক ব্যবহার করে আমরা একটি লাইন অঙ্কন করতে পারি। স্থানাঙ্কসমূহ হল ক্যানভাসের উপর পিক্সেলের অবস্থান। টিকে (Tk) ক্যানভাসে, স্থানাঙ্কসমূহ বলে দেয় ক্যানভাসটি কতটুকু চওড়া হবে (বাম থেকে ডানে) এবং কতটুকু লম্বা হবে (উপর থেকে নিচে)। বাম থেকে ডান অংশটুকুর নাম x-axis এবং উপর নিচ অংশটুকুর নাম y-axis.

যেহেতু ক্যানভাসটি চওড়ায় ৫০০ পিক্সেল এবং উচ্চতায় ৫০০ পিক্সেল, তাই নিম্নে ডান কোণের স্থানাঙ্ক হবে ৫০০, ৫০০। তাই চিত্র ৯.১ এ ০,০ স্থানাঙ্ক থেকে ৫০০,৫০০ স্থানাঙ্কে একটি লাইন অঙ্কন করা যাবে:



চিত্র ৯.১: ক্যানভাস x এবং y এক্সিস

```
>>> from tkinter import*
>>> tk=Tk()
>>> canvas=Canvas(tk, width=500,height=500)
>>> canvas.pack()
>>> canvas.create_line(0,0,500,500)
```

এখন একই জিনিস কাছিমের সাহায্যে করতে হলে আমাদের নিচের কোড প্রয়োজন হবে:

```
>>> import turtle
>>> turtle.setup(width=500,height=500)
>>> t=turtle.Pen()
>>> t.up()
>>> t.goto(-250,250)
>>> t.down()
>>> t.goto(500,-500)
```

সুতরাং ইতোমধ্যে tkinter কোডের উন্নতি সাধিত হয়েছে, আরও ছোট হয়েছে এবং অপেক্ষাকৃত সহজ হয়েছে। ক্যানভাসের জন্য প্রচুর পদ্ধতি রয়েছে, কিছু রয়েছে যা আমাদের ব্যবহার না করলেও চলবে, কিন্তু তারপরও চল আমরা ফাংশনের কিছু আকর্ষণীয় উদাহরণ দেখি।

৯.৩ বক্স অঙ্কন করা

কাছিমকে দিয়ে আমরা বক্স অঙ্কন করার সময়, কাছিমকে সামনে যেতে বলেছি, ঘুরতে বলেছি, আবার সামনে যেতে বলেছি এবং এভাবে একটা বক্সের আকার আমরা পেয়েছিলাম। এভাবে আমরা একটি আয়তকার অথবা বর্গাকার বক্স আঁকতে পারি। tkinter এর সাহায্যে বর্গাকার অথবা আয়তাকার বক্স অঙ্কন করা অপেক্ষাকৃত সহজ-তোমাকে শুধু জানতে হবে কোণের স্থানাঙ্কসমূহ।

```
>>> from tkinter import*
>>> tk=Tk()
>>> canvas=Canvas(tk,width=400,height=400)
>>> canvas.pack()
>>> canvas.create_rectangle(10,10,50,50)
1
```

উপরের উদাহরণে। আমরা একটি ক্যানভাস তৈরি করেছি যা ৪০০ পিক্সেল চওড়া এবং ৪০০ পিক্সেল লম্বা এবং তার উপরের কোণায় আমরা একটি বর্গক্ষেত্র অঙ্কন করেছি (উপরের বামপ্রান্ত উপরে বাম থেকে ১০ পিক্সেল ভিতরে এবং নিচের ডানপ্রান্ত নিচে ডান থেকে ৫০ পিক্সেল ভিতরে)। তুমি হয়ত লক্ষ করে থাকবে যখন তুমি কোড লিখে তা চালিয়েছ তখন একটি নাম্বার দেখিয়েছে। এটা পূর্বে যখন তোমরা create_line কমান্ড দিয়েছিলে তখনও দেখতে পেয়েছিলে। এটা কি? এটা হচ্ছে তুমি কোন আকৃতির ছবি এঁকেছ (বর্গাকার, আয়তাকার, বৃত্তাকার ইত্যাদি) তা সনাক্তকারী। আমরা পরবর্তীতে এই নাম্বার নিয়ে পুনরায় আলোচনা করব।

যেসকল প্যারামিটার আমরা প্রদান করেছি তা হল: উপরে বামপ্রান্তে x অবস্থান, উপরে বামপ্রান্তে y অবস্থান, নিম্নে ডানপ্রান্তে x অবস্থান এবং নিম্নে ডানপ্রান্তে y অবস্থান। বোঝার সুবিধার্থে আমরা এসকল অবস্থানকে x1, y1 এবং x2, y2 ধরে নিতে পারি। আমরা x2 কে বড় সংখ্যা দ্বারা পরিবর্তন করে আয়তক্ষেত্র অঙ্কন করতে পারি:

```
>>> canvas.create_rectangle(100,100,300,50)
```

অথবা y2 কে একটি বড় সংখ্যা ধরে নিতে পারি:

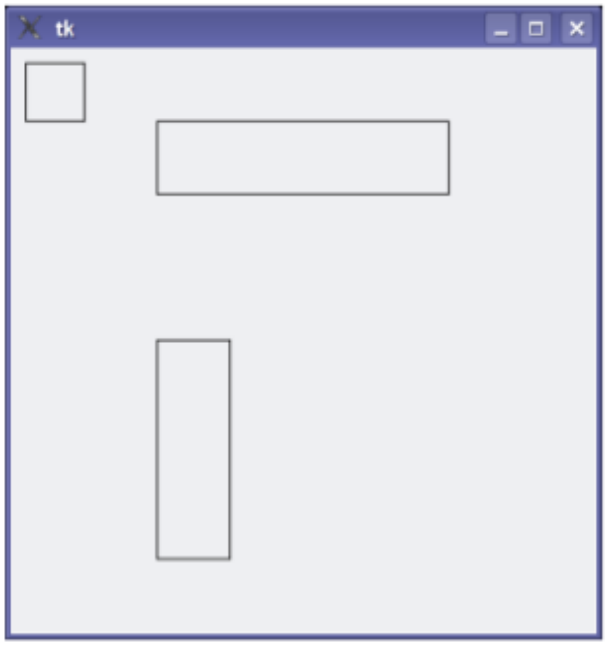
```
>>> canvas.create_rectangle(100,200,150,350)
```

সর্বশেষ আয়তক্ষেত্রটি মূলত বলছে: ক্যানভাসের উপরে বামপ্রান্ত থেকে ১০০ পিক্সেল সামনে যাও, এবং উপর থেকে ২০০ পিক্সেল নিচে যাও, তারপর একটি বক্স অঙ্কন কর ১৫০ পিক্সেল সামনে এগিয়ে এবং ৩৫০ পিক্সেল নিচে নেমে। তাহলে এই মূহুর্তে তোমার ক্যানভাসটি দেখতে চিত্র ৯.২ এর মত হবে।

চল এবার আমরা ক্যানভাসটিকে বিভিন্ন আকৃতির আয়তক্ষেত্র দ্বারা পরিপূর্ণ করি। আমরা random নামক মডিউল ব্যবহার করে এটি করতে পারব। প্রথমে random মডিউল ইমপোর্ট করব:

```
>>> import random
```

তারপর আমরা স্থানাঙ্ক হিসেবে এলোমেলো নাম্বার ব্যবহার করে একটি ফাংশন তৈরি করতে পারব। এখানে যে ফাংশন আমরা ব্যবহার করব তার নাম randrange:



চিত্র ৯.২: tkinter বাক্সসমূহ

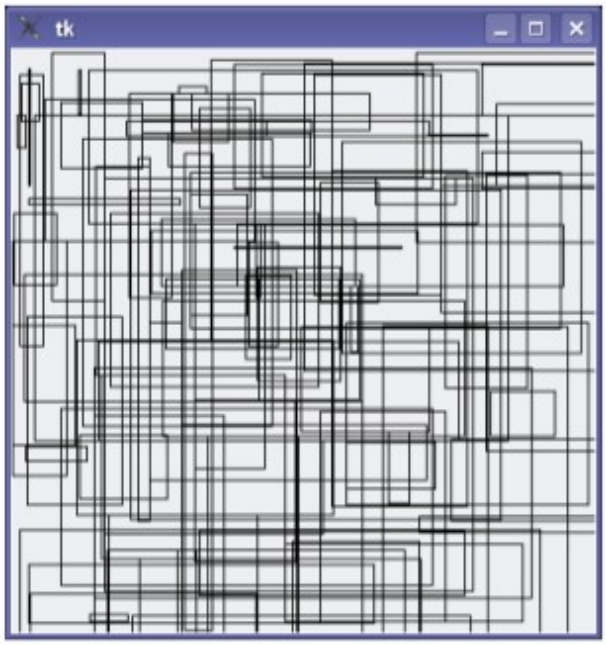
```
>>> def random_rectangle(width,height):
...     x1=random.randrange(width)
…       y1=random.randrange(height)
…       x2=random.randrange(x1+random.randrage(width))
…       y2=random.randrange(y1+random.randrange(height))
…       canvas.create_rectangle(x1,y1,x2,y2)
```

প্রথম দুইলাইনে randrange ব্যবহার করে আয়তক্ষেত্রের উপরে বামপ্রান্তের জন্য আমরা একটি ভ্যারিয়েবল তৈরি করেছি, যেখানে আমরা প্রস্থ এবং উচ্চতা প্রদান করেছি। randrange ফাংশন আর্গুমেন্ট হিসেবে নাম্বার গ্রহণ করে (randrage সম্পর্কে আরও জানতে Appendix C দেখ) তাই randrage(10) তোমাকে ০ থেকে ১০ এর মধ্যে সংখ্যা প্রদান করবে, randrange(100) তোমাকে ০ থেকে ১০০ এর মধ্যে সংখ্যা প্রদান করবে। পরবর্তী দুই লাইন নিম্নে ডানপ্রান্তের আয়তক্ষেত্রের (অথবা বর্গক্ষেত্র!) জন্য ভ্যারিয়েবল তৈরি করে। এখানে আমরা উপরে বামপ্রান্তে (x1 অথবা y1) স্থানাঙ্ক ব্যবহার করেছি এবং এতে এলোমেলো মান যোগ করেছি। সর্বশেষে আমরা create_rectangle ফাংশন কল করেছি এই ভ্যারিয়েবল ব্যবহার করার জন্য। তুমি তোমার ক্যানভাসে প্রস্থ এবং উচ্চতার মান প্রদান

করে random_rectangle ফাংশন ব্যবহারের চেষ্টা করে দেখতে পার:

```
>>> random_rectangle(400,400)
```

অথবা তোমার স্ক্রীনে পরিপূর্ণ করে ফেলতে পার, কেমন হবে যদি, বার বার ফাংশনটি কল করার জন্য আমরা একটি লুপ তৈরি করি?



চিত্র ৯.৩: এলোমেলো অনেকগুলো আয়তক্ষেত্র

```
>>> for x in range(0,100):
…     random_rectangle(400,400)
```

এই কোড ক্যানভাসে বিশৃঙ্খলভাবে আয়তক্ষেত্র তৈরি করে (চিত্র ৯.৩) কিন্তু বিষয়টি মজার।

তোমার কি মনে আছে গত অধ্যায়ে, আমরা কাছিমকে তিনটি রঙের শতকরা মান দিয়েছিলাম (লাল, সবুজ, নীল) আর কাছিমটি অঙ্কন করেছিল? Tkinter ব্যবহার করেও তোমরা এই কাজ করতে পারবে তবে এটি কিছুটা জটিল। প্রথমে আমরা এলোমেলো আয়তক্ষেত্রের ফাংশন পরিবর্তন করব যেন আয়তক্ষেত্র বিভিন্ন রঙ দ্বারা পরিপূর্ণ করা যায়:

```
>>> def random_rectangle(width,height,fill_colour):
…     x1=random.randrange(width)
…     y1=random.randrange(height)
…     x2=random.randrange(x1+random.randrange(width))
…     y2=random.randrange(y1+random.randrange(height))
…     canvas.create_rectangle(x1,y1,x2,y2,fill=fill_colour)
```

create_rectangle ফাংশনটি 'fill' নামক একটি প্যারামিটার গ্রহণ করে যা রঙ পূর্ণ করায় ব্যবহৃত হয়। এখন আমরা এটিকে ফাংশনে প্রদান করব। চেষ্টা করে দেখ:

```
>>> random_rectangle(400,400,'green')
>>> random_rectangle(400,400,'red')
>>> random_rectangle(400,400,'blue')
>>> random_rectangle(400,400,'orange')
>>> random_rectangle(400,400,'yellow')
>>> random_rectangle(400,400,'pink')
>>> random_rectangle(400,400,'purple')
>>> random_rectangle(400,400,'violet')
>>> random_rectangle(400,400,'magenta')
>>> random_rectangle(400,400,'cyan')
```

কিছু এবং সম্ভবত সকল, নামকরণকৃত রঙ কাজ করবে। কিন্তু কিছু রঙ ত্রুটি বার্তা প্রদর্শন করতে পারে(এটা নির্ভর করে তুমি উইন্ডোজ, ম্যাক নাকি লিনাক্স ব্যবহার করছ তার উপর)। এখন পর্যন্ত এটি খুব সহজ লাগার কথা। কিন্তু যদি আমরা স্বর্ণের মত কোন রঙ ব্যবহার করতে চাই তাহলে কি করব? কাছিম মডিউলে আমরা স্বর্ণের রঙ তৈরি করেছিলাম ১০০% লাল, ৮৫% সবুজ এবং কোন নীল রঙ ব্যবহার না করে। tkinter এ আমরা স্বর্ণের রঙ তৈরি করতে পারি যা ব্যবহার করে তা হল:

```
>>> random_rectangle(400,400,'#ffd800')
```

স্বর্ণের রঙ তৈরির জন্য এটা খুব অদ্ভুত একটি উপায়। '#ffd800' এটার নাম হেক্সা ডেসিমাল (hexadecimal) এবং এটা হচ্ছে সংখ্যা প্রকাশের অন্য একটি উপায়। কিভাবে হেক্সা ডেসিমাল নাম্বার কাজ করে তা বর্ণনা করতে গেলে এই বই এর জন্য নির্ধারিত পৃষ্ঠার সংখ্যা বৃদ্ধি পাবে। সুতরাং আপাতত হেক্সা ডেসিমাল রঙ তৈরির জন্য তুমি নিম্নের ফাংশন ব্যবহার করতে পার:

```
>>> def hexcolor(red,green,blue):
…     red=255*(red/100.0)
…     green=255*(green/100.0)
…     blue=255*(blue/100.0)
…     return '#%02x%02x%02x'%(red,gree,blue)'
```

এবার ১০০% লাল, ৮৫ ভাগ সবুজ এবং কোন নীল রঙ ব্যবহার না করে হেক্সা ডেসিমাল স্বর্ণের রঙ তৈরি করি:

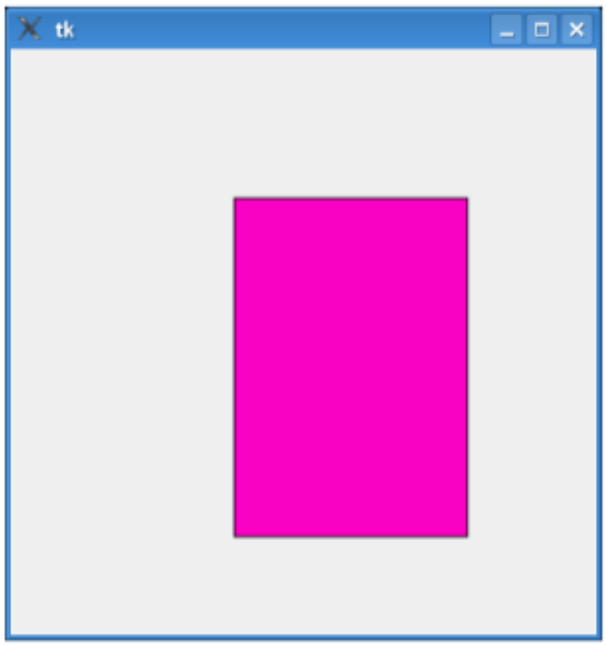
```
>>>  print(hexcolor(100,85,0))
#ffd800
```

তুমি উজ্বল ময়ূরকণ্ঠী বর্ণের রঙ তৈরি করতে পার ৯৮% লাল, ১% সবুজ এবং ৭৭% নীল রঙ ব্যবহার করে:

```
>>>   print(hexcolor(98,1,77))
#f902c4
```

আমরা পূর্বে তৈরি করা এলোমেলো আয়তক্ষেত্র রঙ করার জন্য এগুলো ব্যবহার করতে পারি:

```
>>> random_rectangle(400,400,hexcolor(98,1,77))
```



চিত্র ৯.৪: একটি ময়ূরকণ্ঠী বর্ণের আয়তক্ষেত্র

৯.৪ বৃত্তের পরিধির অংশবিশেষ অঙ্কন করা (আর্ক):

আর্ক হচ্ছে বৃত্তের একটি অংশ, কিন্তু tkinter এর সাহায্যে এটি আঁকতে হলে তোমাকে একটি আয়তক্ষেত্র অঙ্কন করতে হবে। এটা আবার কেমন কথা! কিন্তু যতক্ষণ পর্যন্ত একটি আয়তক্ষেত্রের মধ্যে আর্ক অঙ্কন না করছ ততক্ষণ এটি বোধগম্য হবে না (চিত্র ৯.৫ দেখ)। আর্ক অঙ্কন করার কোডটি হবে এরকম:

```
>>> canvas.create_arc(10,10,200,100,extent=180,style=ARC)
```

এই কোড ক্যানভাসের উপরে বামপ্রান্তে ১০,১০ স্থানাঙ্ক স্থাপন করে (যা ক্যানভাসের ১০ পিক্সেল সামনের দিকে এবং ১০ পিক্সেল নিচে) এবং নিচে ডানপ্রান্তে ২০০,১০০ স্থানাঙ্ক স্থাপন করে (২০০ পিক্সেল সামনে এবং ১০০ পিক্সেল নিচে)। পরবর্তী প্যারামিটার (একটি নামের প্যারামিটার) 'extent' যা আর্কের কোণের ডিগ্রীর জন্য ব্যবহৃত হয়। যদি তুমি বৃত্তের (অথবা আর্কের) ডিগ্রী সম্বন্ধে না জেনে থাক, তাহলে শুধু মনে রাখবে, যদি তুমি একটি বৃত্ত সম্পর্কে ভাব, ১৮০ ডিগ্রী হল তার অর্ধেক (অথবা আর্কের অর্ধেক), ৩৫৯ ডিগ্রী হল একটি পূর্ণ বৃত্ত, ৯০ ডিগ্রী হচ্ছে বৃত্তের এক চতুর্থাংশ এবং ০ ডিগ্রী হচ্ছে....কিছুই না। এখানে কিছু কোড দেয়া হল যা ভিন্ন ভিন্ন আর্ক অঙ্কন করবে যার দ্বারা তুমি ভিন্ন ভিন্ন ডিগ্রীর মৌলিক পার্থক্য বুঝতে সক্ষম হবে। (চিত্র ৯.৬ এ তুমি উদাহরণ দেখতে পার):

চিত্র ৯.৫: আয়তক্ষেত্রের ভেতরে একটি আর্ক

```
>>> canvas.create_arc(10,10,200,80,extent=45,style=ARC)
>>> canvas.create_arc(10,80,200,160,extent=90,style=ARC)
>>> canvas.create_arc(10,160,200,240,extent=135,style=ARC)
>>> canvas.create_arc(10,240,200,320,extent=180,style=ARC)
>>> canvas.create_arc(10,320,200,400,extent=359,style=ARC)
```

৯.৫ উপবৃত্ত অঙ্কন করা

উপরের কোডসমূহের শেষ কোডটি একটি উপবৃত্ত অঙ্কন করে। তুমি create_oval ফাংশন ব্যবহার করেও উপবৃত্ত বা ডিম্বাকৃতির বৃত্ত অঙ্কন করতে পারবে। আর্কের মতই উপবৃত্তকে আয়তক্ষেত্রের ভেতরের সীমানায় অঙ্কন করতে হয়। উদাহরণস্বরূপ, নিচের কোডটি দেখ:

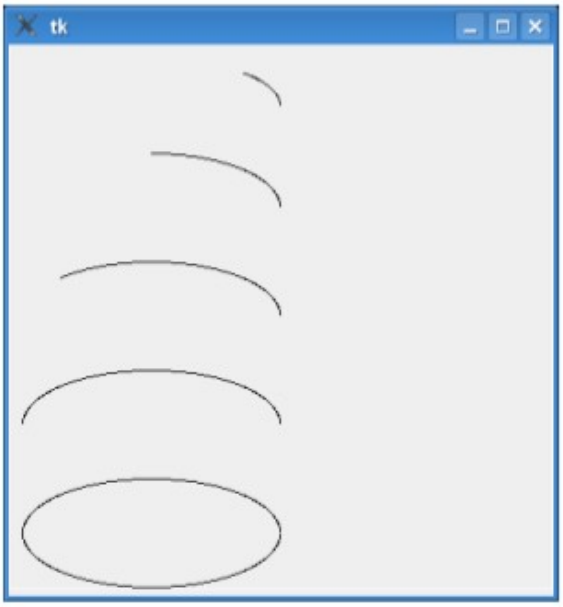
```
>>> tk=Tk()
>>> canvas=Canvas(tk,width=400,height=400)
>>> canvas.pack()
>>> canvas.create_oval(1,1,300,200)
```

এই উদাহরণটি একটি উপবৃত্ত অঙ্কন করে একটি কাল্পনিক বর্গের ভেতরে। যার পিক্সেলের অবস্থান ১,১ থেকে ৩০০,২০০ (চিত্র: ৯.৭)। আমরা যদি একই স্থানাঙ্কে একটি লাল রঙের আয়তক্ষেত্র অঙ্কন করি, তাহলে তোমরা ভালভাবে বুঝবে কিভাবে এর ভেতর উপবৃত্ত অঙ্কন করা হয় (চিত্র: ৯.৮):

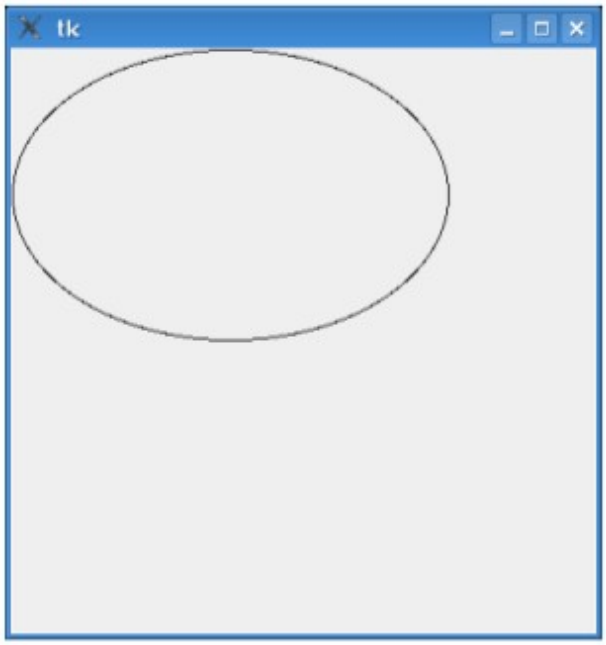
```
>>> canvas.create_rectangle(1,1,300,200,outline="#ff0000")
```

যদি উপবৃত্তের পরিবর্তে একটি পূর্ণ বৃত্ত অঙ্কন করা হয় তাহলে তা থাকবে একটি বর্গের ভেতর (চিত্র: ৯.৯):

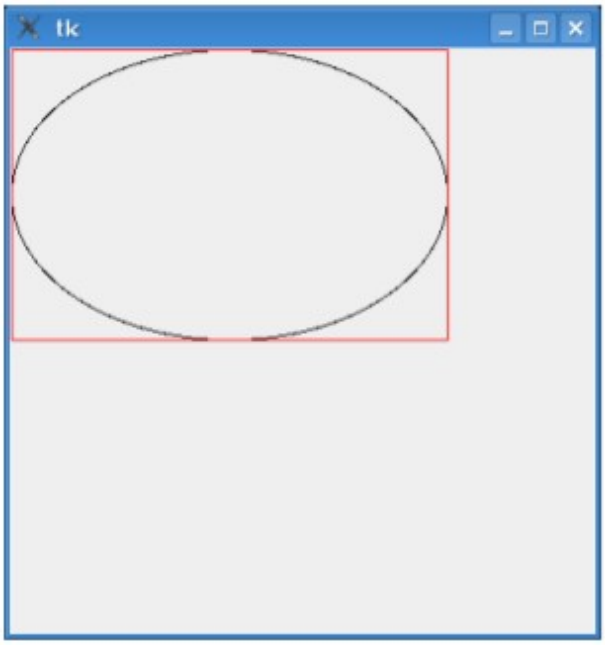
```
>>> tk=Tk()
>>> canvas=Canvas(tk,width=400,height=400)
>>> canvas.pack()
>>> canvas.create_oval(1,1,300,300)
```



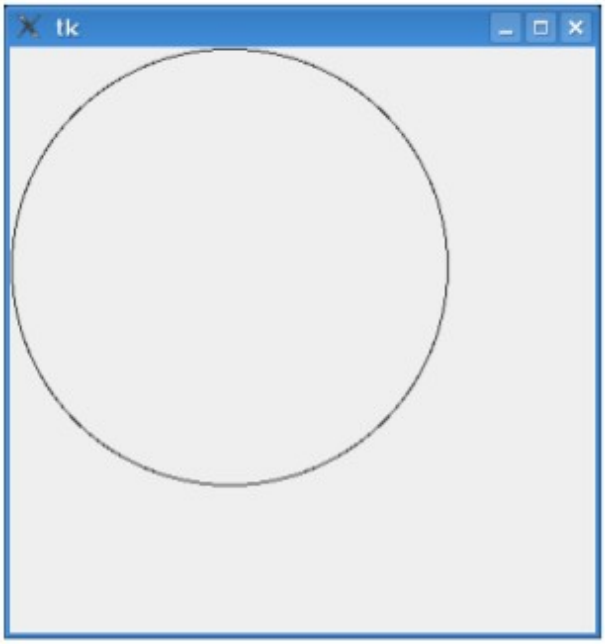
চিত্র ৯.৬: আর্কের বিভিন্ন ডিগ্রী



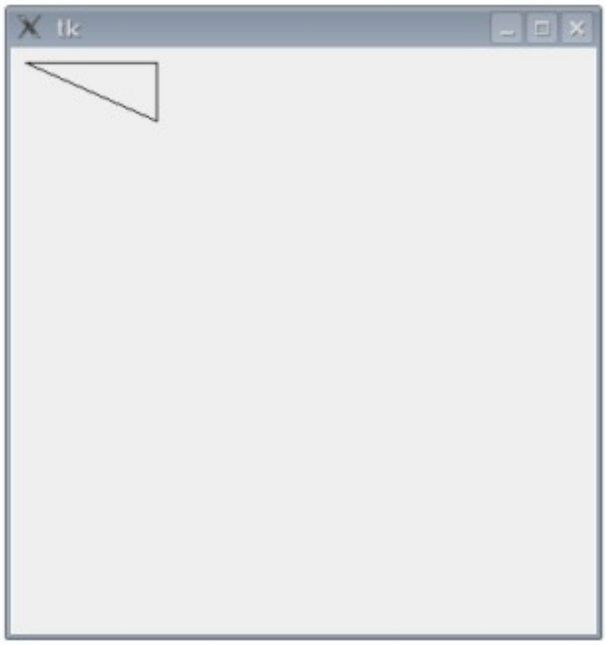
চিত্র ৯.৭: একটি উপবৃত্ত



চিত্র ৯.৮: আয়তক্ষেত্রের ভেতরে উপবৃত্ত



চিত্র ৯.৯: একটি পূর্ণ বৃত্ত



চিত্র ৯.১০: একটি ত্রিভুজ

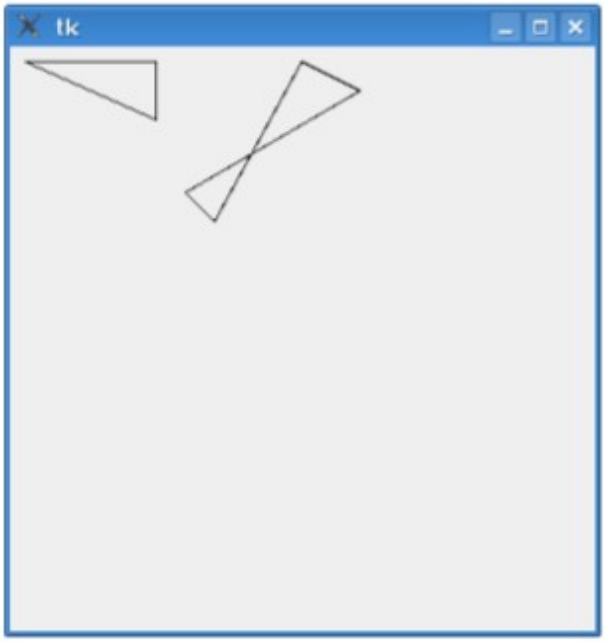
৯.৬ বহুভুজ অঙ্কন করা

বহুভুজ হচ্ছে তিনটি অথবা তার অধিক বাহুবিশিষ্ট যে কোন আকৃতি। ত্রিভুজ, বর্গক্ষেত্র, আয়তক্ষেত্র, পঞ্চভুজ, ষড়ভুজ ইত্যাদি হচ্ছে বহুভুজের উদাহরণ। এর সাথে তুমি চাইলে অনেক অনিয়মিত আকৃতির বহুভুজ তৈরি করতে পারবে। উদাহরণস্বরূপ তুমি যদি একটি ত্রিভুজ অঙ্কন করতে চাও তাহলে তোমাকে তিন সেট স্থানাঙ্ক প্রদান করতে হবে (চিত্র ৯.১০ দেখ):

```
>>> from tkinter import*
>>> tk=Tk()
>>> canvas=Canvas(tk,width=400,height=400)
>>> canvas.pack()
>>> canvas.create_polygon(10,10,100,10,100,50,fill="",outline="black")
```

আমরা চাইলে নিম্নের কোড ব্যবহার করে একটি অনিয়মিত বহুভুজও যোগ করতে পারি। চিত্র ৯.১১ এ দুই ধরণের বহুভুজ আমরা দেখতে পাচ্ছি।

```
>>>canvas.create_polygon(200,10,240,30,120,100,140,120,fill="",out
line="black")
```



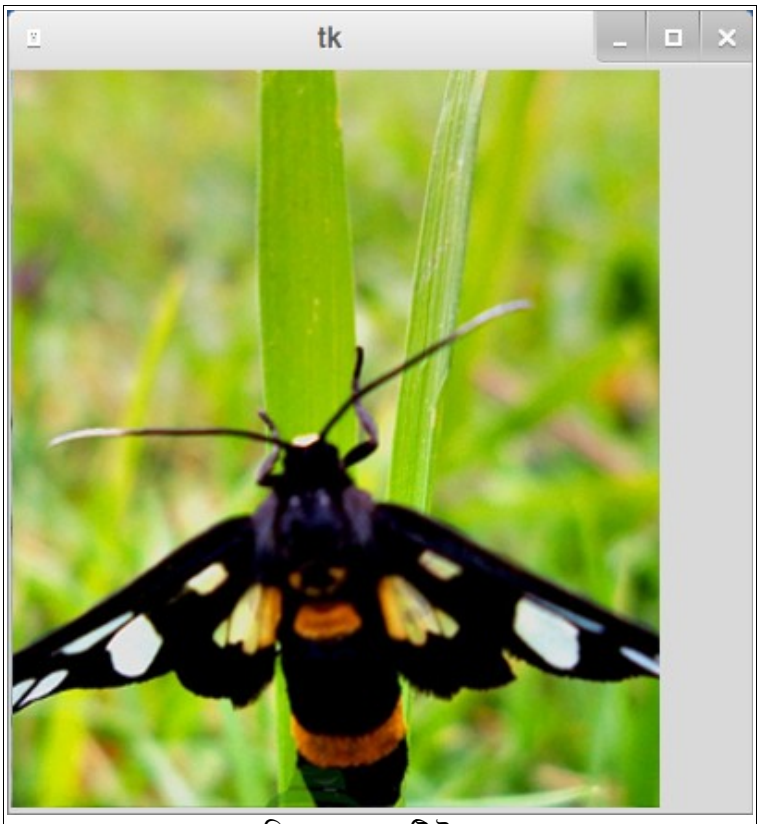
চিত্র ৯.১১: একটি সাধারণ ত্রিভুজ

৯.৭ ইমেজ অঙ্কন করা

তুমি tkinter ব্যাবহার করে ক্যানভাসের উপর ইমেজ অঙ্কন করতে পার এর জন্য প্রথমে ইমেজ লোড করতে হবে, এরপর ক্যানভাস অবজেক্টের উপর create_image ফাংশন ব্যবহার করতে হবে। এটা শুনতে অযৌক্তিক মনে হয়, কিন্তু এটা নিম্নের কোড অনুসারে কাজ করবে:

```
1. >>> from tkinter import*
2. >>> tk=Tk()
3. >>> canvas=Canvas(tk,width=400,height=400)
4. >>> canvas.pack()
5. >>> myimage=PhotoImage(file='test.gif')
6. >>> canvas.create_image(0,0,image=myimage,anchor=NW)
```

লাইন ১ থেকে ৪ এ আমরা আগের মত করেই ক্যানভাস সেট করেছি। ৫ নাম্বার লাইনে myimage নামক ভ্যারিয়েবলে ইমেজটিকে লোড করা হয়েছে। এটা গুরুত্বপূর্ণ যে, তুমি যেই ইমেজটিকে লোড করতে চাও তা অবশ্যই আগে থেকে পাইথন দ্বারা প্রবেশযোগ্য ফোল্ডারে রাখবে। সাধারণভাবে পাইথন কনসোল যে অবস্থান থেকে চলছে সেখানে কোন ফোল্ডারে রাখা হয়। সবচেয়ে ভাল হয় তুমি যদি নিজেই কোথায় ইমেজটিকে রাখবে তা বের করে নাও। এজন্য getcwd() ফাংশন ব্যবহার করতে হবে। কোডটি নিচে দেয়া হল:



চিত্র ৯.১২: একটি ইমেজ

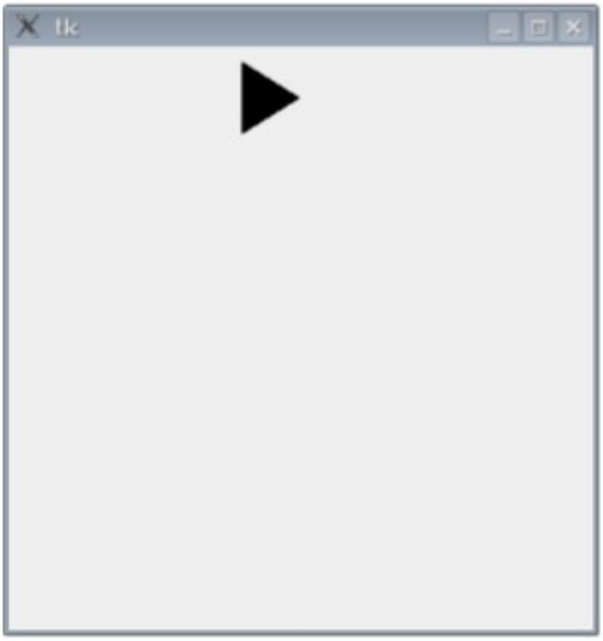
```
>>> import os
>>> print(os.getcwd())
```

এটা সম্ভবত '/home/yourname'... জাতীয় কিছু প্রদর্শন করবে। তখন প্রদর্শিত ডিরেক্টরিতে গিয়ে ইমেজটি রাখবে। অর্থাৎ, তোমার নাম যদি হয় Esha, তাহলে getcwd() তোমাকে দেখাবে '/home/Esha' অথবা '/home/esha' এটা পুরোটাই নির্ভর করছে তোমার কম্পিউটার কীভাবে সেটআপ করেছ এবং কি নাম দিয়ে করেছ তার উপর।

এই ডিরেক্টরিতে তোমার ছবি অনুলিপি কর তারপর PhotoImage ফাংশন দ্বারা তা পাইথনে লোড কর (৫ নাম্বার লাইনের মত)। তারপর তুমি create_image ফাংশন ব্যবহার করে পাইথনের ক্যানভাসে ইমেজটি প্রদর্শন করতে পারবে (৬ নাম্বার লাইন অনুযায়ী)। যদি সবকিছু ঠিক মত কর তাহলে তুমি চিত্র ৯.১২ এর মত কিছু একটা দেখতে পাবে (ছবিটি অবশ্য একইরকম হবে না)।

PhotoImage ফাংশন শুধুমাত্র .gif, .ppm, .pgm এক্সটেনশনের ইমেজ ফাইল লোড করতে পারে। যদি তুমি অন্য ধরনের ইমেজ ফাইল লোড করতে চাও (ইমেজ ফাইল তৈরি করার জন্য অনেক উপায় রয়েছে--উদাহরণস্বরূপ, ডিজিটাল ক্যামেরা যেসকল ছবি তুলে তার এক্সটেনশন হচ্ছে .jpg), তাহলে তোমাকে এক্সটেনশন যোগ করতে হবে

যা পাইথনকে সেই ইমেজটি লোড করার যোগ্যতা প্রদান করবে। The Python Imaging Library (PIL)[১]পাইথনে সকল প্রকার ইমেজ লোড করার যোগ্যতা প্রদান করার সাথে সাথে ইমেজ ছোট বড় করা, ইমেজের রঙ পরিবর্তন করা, ইমেজ উল্টিয়ে দেয়া ইত্যাদি নানাবিধ সুবিধা প্রদান করে থাকে। যদিও, Python Imaging Library ইনস্টল করা এবং তা ব্যবহার করা, এই বইয়ের বিষয়ের অন্তর্ভূক্ত নয়।



চিত্র ৯.১৩: একটি ত্রিভুজ পর্দার চারদিকে ঘুরছে

৯.৮ মৌলিক অ্যানিমেশন

এখন পর্যন্ত, আমরা স্থির চিত্র নিয়ে কাজ করেছি---যা আসলে নড়াচড়া করতে পারে না। এখন একটু অ্যানিমেশন হলে কেমন হয়? অ্যানিমেশন করার জন্য Tk কোন শক্তিশালী স্যুট নয়, তবুও আমরা এর সাহায্যে মৌলিক অ্যানিমেশন করতে সক্ষম হব। উদাহরণস্বরূপ, নিম্নের কোড ব্যবহার করে আমরা একটি রঙে পূর্ণ ত্রিভুজ তৈরি করতে পারি এবং স্ক্রীন এর চারপাশে এটিকে নাড়াচড়া করতে পারি:

```
1. >>> import time
2. >>> from tkinter import*
3. >>> tk=Tk()
4. >>> canvas=Canvas(tk,width=400,height=400)
5. >>> canvas.pack()
6. >>> canvas.create_polygon(10,10,10,60,50,35)
7. 1
8. >>> for x in range(0,60):
9. …     canvas.move(1,5,0)
10.      …    tk.update()
11.      …    time.sleep(0.05)
```

যখন তুমি শেষ লাইনটি লিখে কীবোর্ডের এন্টার চাপবে, ত্রিভুজটি স্ক্রীনের চারপাশে নড়াচড়া করা শুরু করবে (চিত্র ৯.১৩ তে তোমরা দেখতে পাচ্ছ এটি সামনের দিকে অর্ধেকটুকু এগিয়েছে)।

১ The Python Imaging Library পাওয়া যাবে, http://www.pythonware.com/products/pil/index.htm

এটা কীভাবে কাজ করে?

আমরা পূর্বেই দেখেছি, লাইন ১-৫ হচ্ছে একটি ক্যানভাস প্রদর্শনের প্রাথমিক বা মৌলিক সেটআপ এবং লাইন-৬ এ আমরা একটি ত্রিভুজ এঁকেছি (create_polygon ফাংশন ব্যবহার করে)। আর লাইন-৭ এ আছে ফাংশন থেকে প্রাপ্ত সনাক্তকারী (সংখ্যায় লেখা ১)। লাইন-৮ এ ০-৫৯ পর্যন্ত গণনা করার জন্য একটি সাধারণ for-loop সেটআপ করেছি।

লাইন ৯-১১ হচ্ছে ত্রিভুজটিকে স্থানান্তর করার জন্য লিখিত কোড। ক্যানভাসে অবস্থিত বস্তুকে স্থানান্তর করার জন্য বস্তুর x
এবং y নির্দেশকে মান যোগ করতে হয়। এখানে যেমন লাইন-৯ এ ১নং বস্তুকে (ত্রিভূজের সনাক্তকারী) ৫ পিক্সেল সামনের দিকে এবং ০ পিক্সেল নিচের দিকে স্থানান্তরিত করা হলো। পুনরায় পিছনের দিকে স্থানান্তরিত করার জন্য লিখতে হবে canvas.move(1, -5, 0).

অতঃপর tk অবজেক্টের আপডেট ফাংশন একে আপডেট করার জন্য বলবে (যদি আপডেট ফাংশন ব্যবহার করা না হয় তাহলে ত্রিভুজ স্থানান্তরিত হবার পূর্বেই লুপ শেষ হয়ে যাওয়া পর্যন্ত tkinter অপেক্ষা করবে, অর্থাৎ অবজেক্টের স্থানান্তর চোখে পড়বে না)। সবশেষে লাইন-১১ তে অবজেক্ট অনবরত স্থানান্তরের পূর্বে পাইথনকে .০৫ সেকেন্ডের জন্য ঘুমাতে বলা হবে। আমরা যদি move(1, 5, 5) লিখে কোড পরিবর্তন করে দেই তাহলে ত্রিভুজটি পর্দার নিচের দিকে আড়াআড়িভাবে নামবে। ক্যানভাসটি বন্ধ করে (উইন্ডোর উপরের দিকে X বাটনে ক্লিক করে) নিচের কোডটি একবার লিখে দেখা যাক,

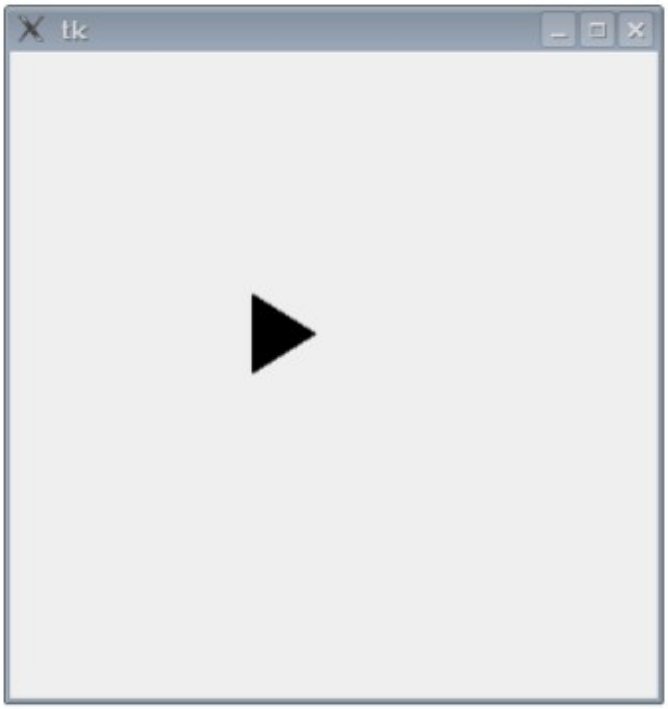
```
>>> i mport time
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_polygon(10, 10, 10, 60, 50, 35)
1

>>> for x in range(0, 60):
...         canvas.move(1, 5, 5)
...         tk.update()
...         time.sleep(0.05)
...
```

চিত্র ৯.১৪ এ আমরা দেখতে পাচ্ছি ত্রিভুজটি পর্দার নিচের দিকে নামছে। -5, -5 লিখে একে আবার পেছনের দিকে ফিরিয়ে নেয়া যায়:

```
>>> import time
>>> for x in range(0, 60):
...         canvas.move(1, -5, -5)
...         tk.update()
...         time.sleep(0.05)
```

৯.৯ রিচিং ইভেন্ট (Reacting to events. . .)

Event Bindings এর মাধ্যমে একটি কী আঘাত করে আমরা ত্রিভুজের প্রতিক্রিয়া বানাতে পারি। প্রোগ্রাম চলার সময় যা ঘটে তাই ইভেন্ট। যেমন: মাউস নাড়ানো, যেকোন কী এর ব্যবহার অথবা কোন উইন্ডো বন্ধ করা। এই



চিত্র ৯.১৪: একটি ত্রিভুজ পর্দার নিচের দিকে নামছে

ইভেন্টগুলো খুঁজে বের করতে Tk সেটআপ করে রাখতে পারো। একটি ফাংশন তৈরি করে ইভেন্টগুলো পরিচালনা শুরু করতে হয়। ধরা যাক, তুমি চাও যে এন্টার কী চাপলে ত্রিভুজটি নড়তে শুরু করবে। এজন্য তুমি একটি ফাংশন তৈরি করতে পারো:

```
>>> def movetriangle(event):
...     canvas.move(1, 5, 0)
```

এখানে ফাংশনের একটি একক প্যারামিটার (ইভেন্ট) থাকতে হবে। Tk এই প্যারামিটার ব্যবহার করে কি হয়েছে তা ফাংশনকে জানাবে। এখন আমরা ক্যানভাসে bind_all ফাংশন ব্যবহার করে Tk-কে বলতে পারবো একটি নির্দিষ্ট ইভেন্টের জন্য এই ফাংশন ব্যবহৃত হবে। কোডটি হবে এইরকম:

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_polygon(10, 10, 10, 60, 50, 35)
>>> def movetriangle(event):
...         canvas.move(1, 5, 0)
...
>>>  canvas.bind_all('<KeyPress-Return>', movetriangle)
```

bind_all ফাংশনের প্রথম প্যারামিটার আমরা যে ইভেন্ট চাচ্ছি তা বর্ণনা করে। আমাদের উল্লেখ্য ইভেন্ট হলো ¡KeyPress-Return¿ (যার মানে হচ্ছে এন্টার কী প্রেস করা)। অর্থাৎ এন্টার কী প্রেস করা ইভেন্ট সংঘটিত হলে movetriangle ফাংশন কাজ করবে। এই কোড দিয়ে কাজ করতে চাইলে মাউস দিয়ে Tk ক্যানভাসে ক্লিক করো এবং কীবোর্ডের এন্টার (অথবা রিটার্ন) কী চাপো।
ভিন্ন ভিন্ন কী চেপে ত্রিভুজের দিক পরিবর্তন করলে কেমন হয়? হতে পারে আলোচ্য কী হবে arrow কী। এজন্য প্রথমে নিচের ত্রিভুজ স্থানান্তরের ফাংশন পরিবর্তন করে লিখতে হবে:

```
>>> def movetriangle(event):
...         if event.keysym == 'Up':
...             canvas.move(1, 0, -3)
...         elif event.keysym == 'Down':
...             canvas.move(1, 0, 3)
...         elif event.keysym == 'Left':
...             canvas.move(1, -3, 0)
...         else:
...             canvas.move(1, 3, 0)
```

movetriangle এর ইভেন্ট অবজেক্টে কিছুসংখ্যক properties[২] থাকে। এদের মধ্যে একটি হচ্ছে keysym যা মূল কী এর একটি স্ট্রিং। যদি keysym এ 'Up' স্ট্রিং থাকে তাহলে (1, 0, -3) প্যারামিটারে canvas.move এর নির্দেশনা দেয়া হবে। আর যদি নিচে যেতে বলা হয় তাহলে প্যারামিটার হবে (1, 0, 3)। মনে রাখতে হবে যে, প্রথম প্যারামিটার হচ্ছে ক্যানভাসে অবস্থানকারী আকৃতির/বস্তুর সনাক্তকারী সংখ্যা, দ্বিতীয় প্যারামিটার হচ্ছে x অক্ষে (আনুভূমিক) যে মান যোগ করতে হবে তা এবং শেষ প্যারামিটারটি হচ্ছে y অক্ষে (উল্লম্ব) যে মান যোগ করতে হবে তা। তাহলে আমরা Tk ফাংশনকে বলতে পারি movetriangle ফাংশন ৪টি ভিন্ন ধরনের কী (উপরে, নিচে, ডানে, বামে) দিয়ে ইভেন্ট পরিচালনা করবে। এখন তাহলে কোডের চেহারা হবে:

```
>>> from tkinter import *
>>> tk = Tk()
>>> canvas = Canvas(tk, width=400, height=400)
>>> canvas.pack()
>>> canvas.create_polygon(10, 10, 10, 60, 50, 35)
1
>>> def movetriangle(event):
...     if event.keysym == 'Up':
...         canvas.move(1, 0, -3)
...         elif event.keysym == 'Down':
...         canvas.move(1, 0, 3)
...     elif event.keysym == 'Left':
...         canvas.move(1, -3, 0)
...     else:
...             canvas.move(1, 3, 0)
...
>>> canvas.bind_all('<KeyPress-Up>', movetriangle)
>>> canvas.bind_all('<KeyPress-Down>', movetriangle)
>>> canvas.bind_all('<KeyPress-Left>', movetriangle)
>>> canvas.bind_all('<KeyPress-Right>', movetriangle)
```

এখন তোমার ত্রিভুজ arrow কী চাপলেই তার নির্দেশনা অনুযায়ী নড়াচড়া করবে।

২ Properties হচ্ছে কোনকিছু বর্ণনাকারী নাম ও মান, যাকে বৈশিষ্ট্য বলা যেতে পারে। যেমন: আকাশকে বর্ণনা করা হয় এর নীল রং দিয়ে। গাড়ির ক্ষেত্রে বলতে পারি এর চাকা আছে। প্রোগ্রামিং এ Properties এর নাম ও মান দুটোই বিদ্যমান।

দশম অধ্যায়

এখন আমরা কোথায় যাবো

স্বাগতম! অবশেষে তুমি এটা তৈরি করতে পেরেছ।

এই বই থেকে তুমি আসলে কী শিখতে পেরেছো? আমার মনে হয় কিছু সাধারণ ধারণা যা তোমাকে অন্যান্য প্রোগ্রামিং শিখতে সহায়তা করবে। পাইথন একটি চমৎকার প্রোগ্রামিং ভাষা, তবে প্রতিটি কাজের জন্য একটি প্রোগ্রামিং ভাষাই সবসময় ভালো হবে তা না। কম্পিউটার প্রোগ্রামিং শেখার জন্য অনেক উপায় তুমি দেখতে পাবে। এগুলো দেখে ভয় পাবার কিছু নেই। কারণ এর অনেকগুলোই আকর্ষণীয়ভাবে তৈরি করা হয়েছে।

যেমন, তুমি যদি গেইমস এর জন্য প্রোগ্রামিং করতে চাও তবে BlitzBasic (www.blitzbasic.com) এর মতো ওয়েব সাইট দেখতে পারো, যেখানে বেসিক প্রোগ্রামিং ভাষা ব্যবহার করা হয়েছে অথবা ফ্ল্যাশ এর কথা বলা যেতে পারে (যা অনেক ওয়েব সাইটে অ্যানিমেশন কিংবা গেইমস এর জন্য ব্যবহার করা হয়, যেমন- Nickelodeon এর ওয়েবসাইট, www.nick.com, যেখানে অজস্র ফ্ল্যাশ ব্যবহার করা হয়েছে)।

তুমি যদি ফ্ল্যাশ প্রোগ্রামিং গেইমস আগ্রহী হও তবে এটি শুরু করার জন্য সবচেয়ে ভালো জায়গা হবে Andy Harris এর লেখা 'Beginning Flash Games Programming for Dummies' বইটি অথবা তুমি যদি আরও বেশি জানতে চাও তবে Serge Melnikov এর 'The Flash 8 Game Developing Handbook' বইটি পড়তো পারো। তুমি যদি ওয়েবে ফ্ল্যাশ গেইম সম্পর্কে জানতে চাও তবে www.amazon.com ঠিকানায় এই সম্পর্কে অসংখ্য বই পাবে।

এছাড়াও আরও কিছু প্রোগ্রামিং এর বই হলো: Jonathon S Harbour এর 'Beginner's Guide to DarkBASIC Game Programming' (এই বইটিতে বেসিক প্রোগ্রামিং ভাষা ব্যবহার করা হয়েছে। Maneesh Sethi এর 'Game Programming for Teens' (এই বইটিতে BlitzBasic ব্যবহার করা হয়েছে)। কিন্তু মনে রাখতে হবে যে, BlitzBasic, DarkBasic and Flash এই তিনটিই আমাদের টাকা দিয়ে কিনতে হবে (অর্থাৎ পাইথনের মতো ফ্রি না)। তাই ঐ প্রোগ্রামিং ভাষাগুলো ব্যবহার করতে চাইলে বাবা-মাকে অবশ্যই জানাতে হবে।

তুমি যদি গেইম তৈরি করার প্রোগ্রামিং ভাষা হিসাবে পাইথনকেই বাছাই করে থাকো তবে www.pygame.org ওয়েব সাইটটি এবং Sean Riley এর লেখা 'Game Programming With Python' এই দুইটি একসাথে কাজে লাগাতে পারবে।

তুমি যদি ঠিক না করতে পারো যে গেইমিং এর জন্য কোন প্রোগ্রামিং ভাষা শিখবে, কিন্তু তুমি যদি পাইথন সম্পর্কে আরো বেশি জানতে চাও তবে Mark Pilgrim এর 'Dive into Python' বইটি দেখতে পারো (www.diveintopython.org বইটি এখানেও পাবে)। এছাড়াও http://docs.python.org/tut/tut.html সাইটিতে অজস্র ফ্রি টিউটোরিয়াল তুমি দেখতে পাবে। এখানে এমন কিছু আছে যা এই বইটিতে আমরা বিস্তারিত আলোচনা করিনি, তুমি সাইটিতে গিয়ে পাইথন সম্পর্কে আরও বিস্তারিতভাবে জানতে পারবে ও পাইথনের সাথে খেলা করতে পারবে।

Good luck and enjoy your programming efforts.
তোমার জন্য শুভ কামনা রইলো এবং আশা করি তুমি তোমার পাইথন প্রোগ্রামিং দক্ষতাকে কাজে লাগাতে পারবে শীঘ্রই।

পরিশিষ্ট A

পাইথনের মূল শব্দসমূহ

অন্যান্য প্রোগ্রামের মতো পাইথনেও কী ওয়ার্ড (key word) গুরুত্বপূর্ণ ভূমিকা পালন করে, যা প্রোগ্রামের নিজস্ব ভাষার দ্বারাই ব্যবহার করা হয়ে থাকে। যদি তুমি এই কীওয়ার্ডগুলোকে ভেরিয়েবল বা অন্য উপায়ে ব্যবহার করার চেষ্টা কর, তবে পাইথন কনসোলে অদ্ভুদ রকমের ত্রুটি (error) সমৃদ্ধ বার্তা দেখতে পারবে। নিচে পাইথন কীওয়ার্ডগুলোর একটি বর্ণনা দেওয়া হলো।

and

একটি স্টেটমেন্ট এ দুইটি এক্সপ্রেশন একত্রিত করতে and ব্যবহার করা হয় (এটা অনেকটা if স্টেটমেন্ট এর মতো), এটা দ্বারা বলা হয় যে স্টেটমেন্ট দুটো অবশ্যই true। উদাহরণস্বরূপ-

```
if age > 10 and age < 20
```

এর দ্বারা বুঝতে পারছি যে বয়স অবশ্যই ১০ এর চেয়ে বড় এবং ২০ এর চেয়ে ছোট হবে।

as

ইমপোর্টেড মডিউলকে অন্য নাম দিতে এই **as** কীওয়ার্ডটি ব্যবহার করা হয়। উদাহরণস্বরূপ-
যদি নিচের উদাহরণের মতো নামসহ আমাদের একটি মডিউল থাকে

```
i_am_a_python_module_that_is_not_very_useful
```

যদি তুমি প্রতিবার এই মডিউলের নাম টাইপ করতে চাও তবে তা তোমার জন্য বিরক্তির কারণ হবে,

```
>>> import i_am_a_python_module_that_is_not_very_useful
>>>
>>>
i_am_a_python_module_that_is_not_very_useful.do_something()
I have done something
>>>
i_am_a_python_module_that_is_not_very_useful.do_something_else()
I have done something else!
```

যখন তুমি এটি ইমপোর্ট করবে তখন একটি নতুন নাম দিতে পারবে, এজন্য শুধুমাত্র একটি নতুন নাম দিলেই চলবে (এটি অনেকটা ডাকনামের মতো):

```
>>> import i_am_a_python_module_that_is_not_very_useful as
notuseful
>>>
>>> notuseful.do_something()
I have done something
>>> notuseful.do_something_else()
I have done something else!
```

যদিও তুমি সম্ভবত as কীওয়ার্ড অনেক বেশি ব্যবহার করতে চাইবে না।

assert

Assert একটি অ্যাডভান্স কীওয়ার্ড। কিছু কোড অবশ্যই সত্যি এটি বোঝাতে Assert ব্যবহার করা হয়। এটি কোডের ত্রুটি ও সমস্যা খুঁজে বের করার একটি উপায়। সাধারণত উচ্চ পর্যায়ের প্রোগ্রামে এটি ব্যবহার করা হয়।

break

কোডের রান করা থামানোর জন্য break ব্যবহার করা হয়। তুমি for-loop এর ভেতরে break নিম্নের মতো করে ব্যবহার করতে পারো:

```
>>> age = 25
>>> for x in range(1, 100):
...     print('counting %s' % x)
...     if x == age:
...     print('end counting')
...      break
```

যদি আমরা একটি ভেরিয়েবল 'age' এর জন্য ১০ নির্ধারণ করি তবে এটির আউটপুট আসবে:

```
counting 1
counting 2
counting 3
counting 4
counting 5
counting 6
counting 7
counting 8
counting 9
counting 10
end counting
```

for-loop সম্পর্কে আরো জানার জন্য পঞ্চম অধ্যায়েটি আবার একটু দেখো।

class

একটি অবজেক্টের ধরন বা type নির্ধারণ করতে class কীওয়ার্ডটি ব্যবহার করা হয়। অনেক প্রোগ্রামিং ল্যাঙ্গুয়েজের জন্য এটি একটি চমৎকার বৈশিষ্ট্য এবং যখন জটিল কোন প্রোগ্রাম লেখা হয় তখন এটি অনেক সহায়তা করে থাকে। কিন্তু এই বইয়ের জন্য এটি অ্যাডভান্স কীওয়ার্ড।

del

কোন কিছুর হাত থেকে মুক্তি পাওয়ার জন্য del একটি গুরুত্বপূর্ণ কীওয়ার্ড। উদাহরণস্বরূপ মনে করো, তুমি তোমার জন্মদিনে কি কি খেলনা চাও তার একটি তালিকা ডায়েরিতে লিখে রেখেছো। কিন্তু কয়েকদিন পর তোমার মনের পরিবর্তন হওয়ায় এই লিস্ট থেকে একটি খেলনা বাদ দিয়ে আরেকটি নতুন খেলনার নাম লিখে রাখতে চাও।

রিমোট কন্ট্রোল গাড়ি
নতুন সাইকেল
~~কম্পিউটার গেইম~~
ভিডিও গেইম

এই তালিকাটিই যদি আমরা পাইথনে লিখি তাহলে তার কোড হবে-

```
>>> what_i_want = ['remote controlled car', 'new bike', 'computer game']
```

এখন আমরা কম্পিউটার গেইমকে বাদ দিয়ে একটি নতুন আইটেম যুক্ত করতে চাই তবে যে কোড লিখতে হবে তা হল:

```
>>> del what_i_want[2]
>>> what_i_want.append('video game')
```

এখন আমরা একটি নতুন তালিকা দেখতে পাবো:

```
>>> print(what_i_want)
['remote controlled car', 'new bike', 'video game']
```

দ্বিতীয় অধ্যায়ে এই বিষয়ে বিস্তারিত পাবে।

elif

if-statement এর একটি অংশ হিসেবে এটি ব্যবহার করা হয়। if অংশে আমাদের এটি দেখতে হবে...

else

এটি প্রায়ই if-statement এর অংশ হিসাবে ব্যবহার করা হয়। if অংশে আমাদের এটি দেখতে হবে...

except

কোডের ত্রুটি নির্ণয়ের জন্য অন্য একটি কীওয়ার্ড except ব্যবহার করা হয়। এটি ব্যবহার করা হয় জটিল প্রোগ্রামগুলোতে, তাই এই বইয়ের জন্য এটি একটি অ্যাডভান্স কীওয়ার্ড।

exec

যদিও exec পাইথন কোডের একটি অংশ, এটি ব্যবহৃত হয় বিশেষ ফাংশন হিসেবে যা অনেকটা স্ট্রিং-এর মতো। উদাহরণস্বরূপ তুমি নিচের মতো করে স্ট্রিং সহ একটি ভেরিয়েবল তৈরি করতে পার:

```
>>> myvar = 'hello there'
```

অতঃপর প্রিন্ট কমান্ড দিলে আমরা রেজাল্ট পাব:

```
>>> print(myvar)
hello there
```

এই স্ট্রিং এর পরিবর্তে কিন্তু তুমি কিছু পাইথন কোড ব্যবহার করতে পার, যেমন:

```
>>> myvar = 'print("hello there")'
```

এবং এরপর তুমি একটি exec ব্যবহার করতে পার, যা স্ট্রিংটিকে একটি ছোটখাট পাইথন প্রোগ্রামে পরিণত করবে

এবং তা রান করাবে:

```
>>> exec(myvar)
hello there
```

এটি একটি অপার্থিব ধারণা এবং কোন কিছু সম্পর্কে আমরা কোন ধারণা তৈরি করি না যতক্ষণ পর্যন্ত না আমাদের তা ব্যবহার করার প্রয়োজন পড়ে। এই exec-ও অনেকটা assert এর মতো একটি অ্যাডভান্স কীওয়ার্ড, যা অনেক জটিল প্রোগ্রামে ব্যবহার করা হয়।

finally

এটি আরেকটি অ্যাডভান্স কীওয়ার্ড। যখন কোন এরর হয় তখন যেন কিছু বিশেষ কোড রান করে তার জন্য এই কীওয়ার্ড ব্যবহার করা হয়। (সাধারণত পিছনে কাজ করা যে কোন ধরনের 'mess' দূর করতে এটি ব্যবহার করা হয়)।

for

for কীওয়ার্ডটি ব্যবহার করা হয় একটি for-loop তৈরি করতে। উদাহরণস্বরূপ;

```
for x in range(0,5):
print('x is %s' % x)
```

উপরের for-loop টি পাঁচবার কোডটিকে পরিচালিত করে ফলাফল তৈরি করে;

```
x is 0
x is 1
x is 2
x is 3
x is 4
```

from

যখন একটি মডিউল ইমপোর্ট করা হয়, তখন from কীওয়ার্ড ব্যবহারের মাধ্যমে পুরো মডিউল না ইমপোর্ট করে আমরা শুধুমাত্র প্রয়োজনীয় অংশটি ইমপোর্ট করতে পারি। উদাহরণস্বরূপ, turtle মডিউলের একটি ফাংশন Pen() যা ব্যবহার করা হয়েছে একটি পেন অবজেক্ট তৈরিতে (মূলত এটি একটি ক্যানভাস যেখানে turtle টি নড়াচড়া করে)। তুমি নিচের মতো করে turtle মডিউল ইমপোর্ট করে pen ফাংশন ব্যবহার করতে পার।

```
>>> import turtle
>>> t = turtle.Pen()
```

অথবা তুমি চাইলে সরাসরি শুধুমাত্র pen ফাংশন ব্যবহার করতে পারো (turtle মডিউলকে রেফার না করেই এটি ব্যবহার করা যায়)।

```
>>> from turtle import Pen
>>> t = Pen()
```

অর্থাৎ তুমি কোন মডিউল ইমপোর্ট না করে সেই মডিউল-এর কোন অংশ ব্যবহার করতে পারবে না। উদাহরণস্বরূপ বলা যায়, time মডিউলের দুইটি ফাংশন localtime এবং gmtime। এখন যদি আমরা localtime ইমপোর্ট করি

এবং gmtime ব্যবহার করার চেষ্টা করি তবে একটি এরর পাবো যেমন:

```
>>> from time import localtime
>>> print(localtime())
(2007, 1, 30, 20, 53, 42, 1, 30, 0)
```

এটা কাজ করবে কিন্তু আমরা যদি এই কোডটি লিখি তবে তা কাজ করবে না:

```
>>> print(gmtime())
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
Name এরর: name 'gmtime' is not defined
```

যেহেতু gmtime কে আমরা নির্দিষ্ট করে দেইনি তাই পাইথন আমাদের জানাবে যে এই gmtime ফাংশনটি কাজ করে না। তবে আমরা একগুচ্ছ ফাংশন একটি মডিউলের ভেতরে ব্যবহার করি এবং তাদের যদি মডিউল নাম না দিয়ে রেফার করতে চাই তবে একটি **asterisk (*):** ব্যবহার করে মডিউলকে ইমপোর্ট করতে পারি:

```
>>> from time import *
>>> print(localtime())
(2007, 1, 30, 20, 57, 7, 1, 30, 0)
>>> print(gmtime())
(2007, 1, 30, 13, 57, 9, 1, 30, 0)
```

এক্ষেত্রে আমরা time মডিউল থেকে সবকিছু ইমপোর্ট করতে পারবো এবং আমরা আলাদা আলাদা নামেও ফাংশনগুলো রেফার করতে পারবো।

global

ষষ্ঠ অধ্যায়ে আমরা scope নিয়ে আলোচনা করেছি। scope হল একটি ভেরিয়েবলের দৃশ্যমানতা বা visibility। যদি একটি ভেরিয়েবলকে আমরা ফাংশনের বাইরে ঘোষণা করি তাহলে সাধারণত এটি ফাংশনের ভেতরে দেখাবে। যদি এটি ফাংশনের ভেতরে ঘোষণা করি তাহলে কিন্তু তা ফাংশনের বাইরে দেখায় না।
global ভেরিয়েবল এই নিয়মের ব্যতিক্রম হিসাবে ব্যবহার করা হয়। যদি আমরা একটি ভেরিয়েবলকে global হিসেবে ঘোষণা করি তবে তা সব জায়গায় দেখা যায়। সুতরাং আমরা global ভেরিয়েবলকে পৃথিবীব্যাপী বা সার্বজনীন ভেরিয়েবল বলতে পারি। যদি আমরা আমাদের পাইথন কনসোলকে একটি ছোট পৃথিবী ভাবি তবে global ভেরিয়েবল সারা পৃথিবীব্যাপি বিস্তৃতি বুঝাবে। উদাহরণস্বরূপ:

```
>>> def test():
...
global a
...
a = 1
...
b = 2
```

কি মনে হয়, যদি প্রথমে print(a) কমান্ড দাও এবং পরে print(b) কমান্ড দাও তবে কি হবে? আমরা যদি ফাংশন দুইটি চালাই তবে দেখতে পাব, প্রথমটি কাজ করছে কিন্তু পরেরটি একটি এরর দেখাচ্ছে।

```
>>> test()
>>> print(a)
1
>>> print(b)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
Name error: name 'b' is not defined
```

আমাদের a ভেরিয়েবলটি এখন global, কিন্তু b ভেরিয়েবলটি শুধুমাত্র ফাংশনের ভেতরে দেখাচ্ছে। মনে রাখবে কোন ভেরিয়বলে মান সংরক্ষণ করার পর তাকে global হিসেবে ঘোষণা দিতে হবে, তার আগে নয়।

if

কোন কিছু সম্পর্কে সিদ্ধান্ত নিতে if ব্যবহার করা হয়, এটি একটি স্টেটমেন্ট। এটি অনেক সময় additional কীওয়ার্ড else এবং elif (else if) এর সাথে ব্যবহার করা হয়। এই স্টেটমেন্টটি "যদি একটি বিষয় সত্যি হয় তবে, একটি নির্দিষ্ট ঘটনা ঘটবে" এই রূপ বলার জন্য ব্যবহার করা হয়। উদাহরণস্বরূপ:

```
if toy_price > 1000:
print('That toy is over-priced')
elif toy_price > 100:
print('That toy is expensive')
else:
print('I would like that toy')
```

উদাহরণে দেখা যাচ্ছে, এখানে if স্টেটমেন্ট বলছে যে যদি খেলনার দাম একহাজার টাকার বেশি হয় তবে তা অতিরিক্ত দামি, যদি পাঁচশ টাকার উপরে দাম হয় তবে তা দামী, অন্যথায় এটি বলতে বলছে, "আমি এই খেলনাটা পছন্দ করি"। চতুর্থ অধ্যায়ে এই বিষয়ে বেশ কয়েকটি উদাহরণ আছে।

import

পাইথনে কাজ করার জন্য কোন মডিউল লোড করার জন্য import কীওয়ার্ড ব্যবহার করা হয় । উদাহরণস্বরূপ:

```
>>> import sys
```

এই কোডটি পাইথনকে বলবে আমরা 'sys' মডিউল ব্যবহার করতে চাই।

in

in কীওয়ার্ডটি ব্যবহার করা হয় এক্সপ্রেশনে, একটি কালেকশন করা আইটেমে একটি নির্দিষ্ট আইটেম আছে কিনা তা খুঁজে বের করতে এটি ব্যবহার করা হয়। উদাহরণস্বরূপ, একটি নম্বরের তালিকা (সংগৃহীত) থেকে 1 নাম্বার খুঁজে বের করতে পারি:

```
>>> if 1 in [1,2,3,4]:
... print('number is in list')
number is in list
```

অথবা আমরা শপিং লিস্টে লেটুস খুঁজে পেতে পারি:

```
>>> shopping_list = [ 'eggs', 'milk', 'cheese' ]
>>> if 'lettuce' in shopping_list:
... print('lettuce is in the shopping list')
```

```
... else:
... print('lettuce is not in the shopping list')

...
lettuce is not in the shopping list
```

is

is কীওয়ার্ড, ইকুয়াল অপারেটরের (=) মতো, যা দুটি বস্তু সমান তা বলতে ব্যবহার করা হয় (উদাহরণস্বরূপ 10==10 ট্রু, 10==11 ফলস)। যা হোক is এবং == এর মাঝে একটি মৌলিক পার্থক্য আছে। তুমি যদি দুইটি জিনিসের মাঝে তুলনা করো, == ট্রু রিটার্ন করবে, যেখানে is সেটা নাও করতে পারে (যদিও তুমি ধারণা করতে পারো জিনিস দুটো সমান)।

যেটা অনেক উচ্চতর প্রোগ্রামিং ধারণা। যা প্রায়ই কনফিউশনে ফেলে, সে ক্ষেত্রে == ব্যবহার করাই শ্রেয়।

lambda

এটি একটি অ্যাডভান্স কীওয়ার্ড। আসলে এটি একটু জটিল। আমরা যদি এটা সম্পর্কে আলোচনা করতে যাই তবে এই বইয়ের পরিধি অনেক অনেক বেড়ে যাবে।

তার চেয়ে এ সম্পর্কে আলোচনা না করাই ভাল।

not

যদি কোন স্টেটমেন্ট সত্যি হয়, তবে not কীওয়ার্ড এটাকে মিথ্যা বানাতে পারে। যদি আমরা একটি ভেরিয়েবল x তৈরি করি এবং এর মান নির্ধারণ করি true...

```
>>> x = True
```

...এবং আমরা যদি এই x এর মান not ব্যবহার করে প্রিন্ট করি তাহলে আমরা যা পাবো তা হল:

```
>>> print(not x)
False
```

if-statements-এ ব্যবহার করার পূর্ব পর্যন্ত এই কিওয়ার্ড খুব একটা দরকারি হয় না।
উদাহরণস্বরূপ, যদি তোমার বয়স ১২ বছর হয় এবং ১২ বছর তোমার জন্য খুবই গুরুত্বপূর্ণ একটি বছর হয় এবং তুমি আর কোন বছরকে রেফার করতে চাও না, তাহলে তুমি যা লিখবে:

```
"1 is not an important age" "2 is not an important age" "3 is
not an important age" "4 is not an important age" ... ... "50
is not an important age"
```

এবং এভাবে আরও অনেকবার,
if-statements এর শর্ত অনুসারে লিখতে পারি...

```
if age == 1:
print("1 is not an important age")
elif age == 2:
```

```
print("2 is not an important age")
elif age == 3:
print("3 is not an important age")
elif age == 4:
print("4 is not an important age")
```

এভাবে চলতেই থাকবে...তবে এটি লেখার একটি সাধারণ উপায় আছে:

```
if age < 10 or age > 10:
print("%s is not an important age" % age)
```

কিন্তু এর চেয়েও সহজ একটি উপায় হলো, not এর দ্বারা if-statements ব্যবহার করা:

```
if not age == 10:
print("%s is not an important age" % age)
```

এতক্ষণে তুমি হয়তো বুঝতে পেরেছো, “if age is not 10” বলার আরেকটি সহজ উপায়।

or

একটি স্ট্রেটমেন্টের ভেতরের দুইটি এক্সপ্রেশনকে জোড়া দেওয়ার জন্য or কীওয়ার্ডটি ব্যবহার করা হয়(অনেকটা if-statements এর মতো)। এর দ্বারা বলা হয় যে এর যেকোন একটি এক্সপ্রেশন সত্যি। উদাহরণস্বরূপ:

```
>>> if friend == 'Rasel' or friend == 'Robi':
...
print('The Robin')
... elif friend == 'Belal' or friend == 'Bobi':
...
print('The Babita')
```

এই ক্ষেত্রে, যদি ভেরিয়েবল রাসেল কিংবা রবি খুঁজে পায় তাহলে ‘The Robin’ প্রিন্ট করবে। যদি ভেরিয়েবল বেলাল এবং ববি খুঁজে পায় তবে 'The Babita' প্রিন্ট করবে।

pass

অনেক সময় তুমি যখন একটি প্রোগ্রাম লিখো তখন তুমি এর অংশ বিশেষ লিখে থাকো এবং এর ফলাফল দেখতে চেষ্টা করো। সমস্যা তখনই হবে যখন তুমি একটি if-Statement লিখবে কিন্তু তার কোড শেষ না করেই ফলাফল দেখতে চেষ্টা করবে। তুমি অবশ্যই for-loop পাবে না যদি 'for-loop'-এর কোড ব্লক শেষ করো।

```
>>> if age > 10:
... print('older than 10')
```

এই কোডটি কাজ করবে, কিন্তু তুমি যদি টাইপ কর:

```
>>> if age > 10:
...
```

তাহলে তুমি নিচের মতো করে পাইথন কনসোলে একটি এরর মেসেজ দেখতে পাবে:

```
File "<stdin>", line 2
^
Indentation error: expected an indented block
```

তুমি যদি কোড শেষ করার জন্য এই জাতীয় স্টেটমেন্ট লিখে থাক তবে পাইথন এই জাতীয় এরর মেসেজ দিবে। pass কীওয়ার্ড এই ক্ষেত্রে ব্যবহার করা হয়, যাতে তুমি একটি স্টেটমেন্ট লিখতে পার, কিন্তু যদি কোডটি ব্লক করে রাখো তবে এটি কাজ করবে না। উদাহরণস্বরূপ, তুমি একটি পাইথন কোড লিখবে এবং তাতে তুমি for-loop ব্যবহার করবে। এটি করবে if-statement এর ভেতরে। কিন্তু তুমি এখনও ঠিক কর নাই, if-statement এ কি রাখবে। তুমি হয়তো print করতে পারো আবার break রাখতে পারো অথবা অন্য কিছুও করতে পারো। এই সব ক্ষেত্রে তুমি pass কীওয়ার্ড ব্যবহার করতে পারো এবং তখন কোডটি কাজ করবে (এমনকি তুমি কি করতে চাইছো তা নাও হতে পারে)। কোডটি হল:

```
>>> for x in range(1,7):
... print('x is %s' % x)
... if x == 5:
... pass
```

ফলাফল হিসাবে পাবে:

```
x is 1
x is 2
x is 3
x is 4
x is 5
x is 6
```

পরবর্তীতে তুমি pass এর বদলে if-statement এর কোড ব্লকটি বসাতে পারবে।

print

print কীওয়ার্ড পাইথন কনসোলে কোন স্ট্রিং, নম্বর কিংবা ভেরিয়েবল লিখে থাকে। যেমন:

```
print('hello there')
print(10)
print(x)
```

raise

এটি একটি অ্যাডভান্স কীওয়ার্ড। এক্ষেত্রে raise ব্যবহার করা হয় একটি এরর সংগঠিত হওয়ার কারণ হিসাবে। যা মনে হয় কোন অদ্ভুত ঘটনা, তবে অ্যাডভান্স প্রোগ্রামে এটি খুব দরকারী একটি কীওয়ার্ড।

return

return কীওয়ার্ড ব্যবহার করা হয় কোন ফাংশনকে কোন মান ফিরিয়ে দেওয়ার জন্য। উদাহরণস্বরূপ, তুমি একটি ফাংশন তৈরি করতে পারো যা তুমি যে টাকা জমা রেখেছিলে তার থেকে কিছু ফেরত দেয়:

```
>>> def mymoney():
... return money_amount
```

যখন তুমি এই ফাংশনকে কল করবে তখন এর মানটি অন্য একটি ভেরিয়্যাবলের মান ফেরত দিবে।

```
>>> money = mymoney()
```

অথবা প্রিন্ট করতে পারে,

```
>>> print(mymoney())
```

try

try কীওয়ার্ড একটি ব্লক কোডের শুরুর অংশ যা শেষ হয় except এবং/অথবা finally কীওয়ার্ড দ্বারা। সবকিছু একসাথে অর্থাৎ try/except/finally ব্যবহার করা হয় একটি প্রোগ্রামের এরর মেসেজ নিয়ন্ত্রণ করার জন্য। উদাহরণস্বরূপ, এই প্রোগ্রাম ব্যবহারকারীর জন্য অবশ্যই একটি উপকারী মেসেজ প্রদর্শন করবে একটি খারাপ পাইথন এরর-এর স্থলে।

while

একটি for-loop এর মতো while হচ্ছে লুপিং কোডের অন্য একটি উপায়। যেখানে for-loop কোড একটি সীমানা পর্যন্ত কাউন্ট করে সেখানে while একটি এক্সপ্রেশন সত্য হওয়া পর্যন্ত চলতে থাকে। তাই তোমাকে while ব্যবহার করার সময় সতর্ক থাকতে হবে। কারণ একটি এক্সপ্রেশন যদি সবসময় সত্য হয় তবে while সবসময় চলতেই থাকবে, কখনও বন্ধ হবে না (এটাকে সীমাহীন লুপ বলা হয়)। উদাহরণস্বরূপ:

```
>>> x = 1
>>> while x == 1:
... print('hello')
```

যদি তুমি ওপরের এই কোডটি চালাও তবে তা চিরদিনের জন্য চলতে থাকবে, যতক্ষণ না তুমি পাইথন কনসোল বন্ধ করবে অথবা CTRL+C চাপো (একত্রে control কী এবং C কী) এটিকে বন্ধ করতে পারে। যাহোক while এর কোডটি নিম্নরূপ:

```
>>> x = 1
>>> while x < 10:
... print('hello')
... x = x + 1
```

যদি তুমি নয়বার hello প্রিন্ট করতে চাও (প্রতিবার x চলকের সাথে 1 যোগ হবে যতক্ষণ না x, 10 এর চেয়ে বড় হয়) এটি অবশ্যই একটি for-loop এর মতো, কিন্তু এটি একটি সুনির্দিষ্ট অবস্থায় চলে।

with

এটি একটি অ্যাডভান্স কীওয়ার্ড।

yield

এটি আরেকটি অ্যাডভান্স কীওয়ার্ড।

পরিশিষ্ট B

পূর্বনির্ধারিত ফাংশনগুলো

পাইথনে অনেক পূর্বনির্ধারিত ফাংশন আছে- এই ফাংশনগুলো ব্যবহার করার পূর্বে ইমপোর্ট করার প্রয়োজন নেই। নিচে কিছু পূর্বনির্ধারিত ফাংশনের বর্ণনা দেয়া হল।

abs

abs ফাংশনটি কোন নম্বরের মানকে পরম মান হিসাবে রিটার্ন করে। পরম মান হলো, এমন মান যা ঋনাত্মক নয়। যেমন আমরা বলতে পারি ১০ এর পরম মান ১০ এবং -২০.৫ এর পরম মান ২০.৫০। উদাহরণস্বরূপ বলা যেতে পারে:

```
>>> print(abs(10))
10
>>> print(abs(-20.5))
20.5
```

bool

কোন প্যারামিটারের মানের উপর ভিত্তি করে bool ফাংশন true অথবা false রিটার্ন করে। নম্বরের জন্য 0 রিটার্ন করে false হিসেবে, যেখানে অন্য যে কোন নম্বর true হিসেবে রিটার্ন করে।

```
>>> print(bool(0))
False
>>> print(bool(1))
True
>>> print(bool(1123.23))
True
>>> print(bool(-500))
True
```

অন্য মানের জন্য, যতক্ষণ না কোনকিছু true হয় ততক্ষণ কোন কিছুকে false হিসাবে রিটার্ন করে না।

```
>>> print(bool(None))
False
>>> print(bool('a'))
True
```

cmp

যদি দুইটি মানের মাঝে প্রথমটি দ্বিতীয়টি থেকে তুলনামূলকভাবে ছোট হয় তবে cmp ফাংশন ঋণাত্মক মান রিটার্ন করে। যদি প্রথম মানটি দ্বিতীয়টির সমান হয় তবে 0 রিটার্ন করে এবং যদি প্রথমটি দ্বিতীয়টির থেকে বড় হয় তবে ধনাত্মক মান রিটার্ন করে। উদাহরণস্বরূপ:
২ এর চেয়ে ১ ছোট হয় তবে:

```
>>> print(cmp(1,2))
-1
```

যদি ২ এর সমান ২ হয় তবে:

```
>>> print(cmp(2,2))
0
```

কিন্তু ১ থেকে ২ বড় হয় তবে:

```
>>> print(cmp(2,1))
1
```

তুলনা যে শুধুমাত্র সংখ্যার সাথে হবে তাই না, অন্যান্য মান যেমন স্ট্রিং এর ক্ষেত্রেও হতে পারে:

```
>>> print(cmp('a','b'))
-1
>>> print(cmp('a','a'))
0
>>> print(cmp('b','a'))
1
```

কিন্তু স্ট্রিং নিয়ে কাজ করার সময় সাবধানতা অবলম্বন করতে হবে, অনেক সময় রিটার্ন করা মান তুমি যা আশা করো তার মতো নাও হতে পারে...

```
>>> print(cmp('a','A'))
1
>>> print(cmp('A','a'))
-1
```

ছোট হাতের a আসলে বড় হাতের হরফ A থেকে বড়, অবশ্যই...

```
>>> print(cmp('aaa','aaaa'))
-1
>>> print(cmp('aaaa','aaa'))
1
```

... তিন অক্ষরের a (aaa) চার অক্ষরের a (aaaa) থেকে ছোট।

dir

একটি মান সম্পর্কে যাবতীয় তথ্য dir ফাংশন রিটার্ন করে। তুমি dir ফাংশনটি ব্যবহার করতে পারো যেকোন স্ট্রিং, নম্বর, ফাংশন, মডিউল, অবজেক্ট, ক্লাশ-যে কোন ক্ষেত্রে। কিছু মানের ক্ষেত্রে তথ্য খুব একটা দরকারী নাও হতে পারে (প্রকৃত পক্ষে এটা অনেক কিছু বোঝায় না)। উদাহরণস্বরূপ, ১ নম্বর ফলাফলের জন্য যদি আমরা dir ফাংশন কল করি তাহলে যা পাবো.....

```
>>> dir(1)
['__abs__',  '__add__',  '__and__',  '__class__',  '__cmp__',
'__coerce__', '__delattr__',
'__div__', '__divmod__', '__doc__', '__float__', '__floordiv__',
'__getattribute__',
```

```
'__getnewargs__', '__hash__', '__hex__', '__index__', '__init__',
'__int__', '__invert__',
'__long__', '__lshift__', '__mod__', '__mul__', '__neg__',
'__new__', '__nonzero__',
'__oct__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdiv__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__',
'__rfloordiv__',
'__rlshift__', '__rmod__', '__rmul__', '__ror__', '__rpow__',
'__rrshift__', '__rshift__',
'__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__str__',
'__sub__', '__truediv__',
'__xor__']
```

একটি বড় নম্বরের ফাংশন বাদ দিয়ে আমরা যদি a ফাংশনের জন্য dir কলিং করি তবে রেজাল্ট হিসাবে পাবো.....

```
>>> dir('a')
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__',
'__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__',
'__str__', 'capitalize', 'center', 'count', 'decode', 'encode',
'endswith', 'expandtabs',
'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',
'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

ফাংশন যদি বড় হরফের হয় তবে আমরা দেখতে পাই, (কোন স্ট্রিং এর প্রথম হরফ পরিবর্তন করে বড় হাতের কর).....

```
>>> print('aaaaa'.capitalize())
Aaaaa
```

...isalnum (যদি একটি স্ট্রিং অ্যালফানিউমেরিক হয় তবে true রিটার্ন করে-এটা শুধুমাত্র অক্ষর এবং নম্বর ধারণ করে), isalpha (যদি একটি স্ট্রিং শুধুমাত্র অক্ষর ধারণ করে তবে true রিটার্ন করে) এবং আরও কিছু। যখন তোমার একটি ভেরিয়েবল থাকে এবং তুমি যা করতে চাও তা দ্রুত খুঁজে বের করতে চাও তাহলে dir খুবই উপকারি হতে পারে।

eval
eval ফাংশনকে প্যারামিটার হিসাবে গ্রহণ করে থাকে এবং এমনভাবে রান করে যেন মনে হয় এটি একটি পাইথন এক্সপ্রেশন। এটি অনেকটা exec কীওয়ার্ড এর মতো, কিন্তু কাজ করে কিছুটা ভিন্ন উপায়ে। exec দিয়ে তুমি একটি স্ট্রিং এর ভেতর ছোটখাট পাইথন প্রোগ্রাম তৈরি করতে পারো কিন্তু eval শুধুমাত্র তোমাকে সাধারণ এক্সপ্রেশন হিসাবে অনুমতি দিবে, যেমন:

```
>>> eval('10*5')
50
```

file
এই ফাংশনটি একটি ফাইল তৈরি করা ও এর ভেতরের অবজেক্টগুলো রিটার্ন করা ও ফাইলের ভেতরের তথ্যগুলোতে (ফাইলের কনটেন্ট, আকার এবং আরও কিছুতে) এক্সেসের অনুমতি দেয়। তুমি ফাইল এবং ফাইল অবজেক্ট সম্পর্কে সপ্তম অধ্যায়ে বিস্তারিত জানতে পারবে।

float
float ফাংশন একটি স্ট্রিং-কে অথবা একটি floating point-কে নম্বরে রূপান্তর করে। একটি floating point নম্বর হলো এমন একটি নম্বর যা দশমিক স্থানসহ থাকে। যাকে একটি সত্যিকারের নম্বর বলা হয়।

উদাহরণস্বরূপ, 10 হল একটি integer (যদিও পুরো নম্বর হিসেবে বলা হয়)। কিন্তু এর মাঝেও 10.0, 10.1, 10.253, থাকে এবং এদের সবাইকে দশমিক সংখ্যা (floats) বলা হয়। তুমি একটি স্ট্রিং-কে floating এ পরিণত করতে পারো যেভাবে:

```
>>> float('12')
12.0
```

তুমি একটি স্ট্রিং-এর ভেতরে যেভাবে দশমিক স্থান ব্যবহার করতে পারো:

```
>>> float('123.456789')
123.456789
```

তুমি যেভাবে একটি নম্বরকে float বানাতে পারো:

```
>>> float(200)
200.0
```

অবশ্যই তুমি একটি floating কে আরেকটি floating point এ পরিণত করতে পারো:

```
>>> float(100.123)
100.123
```

তুমি যদি float-কে কল কর তবে যা পাবে 0.0.

int
int ফাংশনটি একটি স্ট্রিং অথবা নম্বরকে একটি পুরো নম্বরে রূপান্তর করে (অথবা integer)।
উদাহরণস্বরূপ:

```
>>> int(123.456)
123
>>> int('123')
123
```

এটি প্রায় অনেকটা float ফাংশনের মতোই কাজ করে থাকে। তুমি যদি একটি floating point নম্বরকে স্ট্রিং-এ রূপান্তর করতে চাও তবে একটি এরর বার্তা দেখবে:

```
>>> int('123.456')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '123.456'
```

যা হোক, তুমি যদি একটি int কে কোন আর্গুমেন্ট ছাড়াই ডাকো তবে তুমি 0 ফেরত পাবে।

len

len ফাংশন একটি অবজেক্টের length-কে রিটার্ন করে। একটি স্ট্রিংয়ের বেলায় এটি স্ট্রিংয়ের নম্বরগুলো রিটার্ন করে।

```
>>> len('this is a test string')
21
```

তালিকা অথবা টাপলের (tuple) এর ক্ষেত্রে এটি আইটেমের নম্বর রিটার্ন করে:

```
>>> mylist = [ 'a', 'b', 'c', 'd' ]
>>> print(len(mylist))
4
>>> mytuple = (1,2,3,4,5,6)
>>> print(len(mytuple))
6
```

একটি ম্যাপের ক্ষেত্রে এটি সর্বদাই তালিকার নম্বর রিটার্ন করে:

```
>>> mymap = { 'a' : 100, 'b' : 200, 'c' : 300 }
>>> print(len(mymap))
3
```

তুমি এই len ফাংশনটির সবচেয়ে বেশি ব্যবহার পাবে loops এর ক্ষেত্রে। তুমি যদি একটি তালিকার উপাদানকে গুণতে চাও তাহলে নিচের কোডটি ব্যবহার করতে পারো:

```
>>> mylist = [ 'a', 'b', 'c', 'd' ]
>>> for item in mylist:
... print(item)
```

যা তালিকার (a, b, c, d) সকল উপাদানকে প্রিন্ট করে থাকে-কিন্তু তুমি যদি তালিকার প্রতিটি উপাদানকে সূচির মতো করে চাও তবে কি হবে? এই ক্ষেত্রে আমরা তালিকার length দেখতে পারি এবং সেখান থেকে তালিকা গণনা করতে পারি।

```
>>> mylist = [ 'a', 'b', 'c', 'd' ]
>>> length = len(mylist)
>>> for x in range(0, length):
...  print('the item at index %s is %s' % (x, mylist[x]))
...
the item at index 0 is a
the item at index 1 is b
the item at index 2 is c
the item at index 3 is d
```

আমরা তালিকার length কে একটি ভেরিয়েবল length দ্বারা সংরক্ষণ করতে পারি এবং আমরা এই ভেরিয়েবলটি আমাদের লুপ তৈরির সময় রেঞ্জ ফাংশনে ব্যবহার করতে পারি।

max

একটি তালিকার, টাপলের কিংবা একটি স্ট্রিংয়ের সর্বোচ্চ উপাদান রিটার্ন করে এই max ফাংশন। উদাহরণস্বরূপ:

```
>>> mylist = [ 5, 4, 10, 30, 22 ]
>>> print(max(mylist))
30
```

একটি স্ট্রিংয়ের উপাদানগুলো যদি কমা দ্বারা পৃথক থাকে অথবা খালি স্থান দ্বারা পৃথক থাকে তবে এটি কাজ করবে:

```
>>> s = 'a,b,d,h,g'
>>> print(max(s))
h
```

এবং তোমার যদি একটি তালিকা টাপল বা স্ট্রিং নাও থাকে, তারপরও ফাংশনের ভেতরে সরাসরি max ব্যবহার করে কোন নম্বরের আর্গুমেন্ট থেকে বড়টা কল করতে পারবে।

```
>>> print(max(10, 300, 450, 50, 90))
450
```

min

max এর মতো একইভাবে min ফাংশনটি কাজ করে থাকে। তবে এটি তালিকা/টাপল/স্ট্রিং থেকে সবচেয়ে ছোটটি কে রিটার্ন করে।

```
>>> mylist = [ 5, 4, 10, 30, 22 ]
>>> print(min(mylist))
4
```

range

রেঞ্জ ফাংশনটি প্রধানত for-loop এর ক্ষেত্রেই ব্যবহার করা হয়, যখন আমি চাইবো কিছু নম্বরের কোডসহ পুনরাবৃত্ত হোক। আমরা সর্বপ্রথম পঞ্চম অধ্যায়ে range নিয়ে আলোচনা করেছি, তখন দেখেছি কি করে এটি দুইটি আর্গুমেন্ট এ ব্যবহার করা হয়। কিন্তু এটি তিনটি আর্গুমেন্টেও ব্যবহার করা যায়। নিচে দুই আর্গুমেন্টের কিছু উদাহরণ দেওয়া হল:

```
>>> for x in range(0, 5):
...
```

```
print(x)
...
0
1
2
3
4
```

তুমি হয়তো যেটা বুঝতে পারবে না যে, রেঞ্জ ফাংশন সত্যিকারের অর্থে স্পেশাল অবজেক্টকে রিটার্ন করে (একে iterator বলে)। যা ফর-লুপের মাধ্যমে কাজ করে। তুমি ইটেরিটর (iterator) কে তালিকায় রূপান্তর করতে পারো (যা যথেষ্ট বিরক্তিকর, এরচেয়ে লিস্ট ফাংশন ব্যবহার কর)। অতএব তুমি যদি রেঞ্জকে কল করে মানের রিটার্ন চাও বা প্রিন্ট কর, তবে এর ভেতর ধারণ করা নম্বর দেখতে পারবে।

```
>>> print(list(range(0, 5)))
[0, 1, 2, 3, 4]
```

তুমি একটি তালিকা পাবে যা ভেরিয়েবল হিসেবে অ্যাসাইন করা হয়েছিল এবং তোমার প্রোগ্রামে elsewhere হিসেবে ব্যবহার করা হচ্ছে:

```
>>> my_list_of_numbers = list(range(0, 30))
>>> print(my_list_of_numbers)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]
```

রেঞ্জ সর্বদা তৃতীয় আর্গুমেন্ট গ্রহণ করে থাকে, যাকে step বলা হয় (প্রথম দুইটি আর্গুমেন্ট হলো start এবং stop)। যদি step এর মান ফাংশনে অতিবাহিত না হয় (অন্যভাবে যখন তুমি শুধুমাত্র start এবং stop কল করবে) তাহলে পূর্বনির্ধারিতভাবে ১ ব্যবহার করা হবে। কিন্তু কি হবে যখন তুমি step এ ২ ব্যবহার করবে? নিচের উদাহরণ দেখো:

```
>>> my_list_of_numbers = list(range(0, 30, 2))
>>> print(my_list_of_numbers)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

তালিকার প্রতিটি নম্বর পূর্বের চেয়ে ২ করে বেশি বাড়ছে। আমরা বৃহৎ step ব্যবহার করতে পারি:

```
>>> mylist = list(range(0, 500, 50))
>>> print(mylist)
[0, 50, 100, 150, 200, 250, 300, 350, 400, 450]
```

এটি ০ থেকে ৫০০ পর্যন্ত একটি তালিকা তৈরি করে (কিন্তু এখানে অবশ্যই ৫০০ থাকবে না), প্রতিবার ৫০ করে বৃদ্ধি ঘটে।

sum

sum ফাংশন কোন তালিকায় উপাদান যোগ করে এবং একটি সম্পন্ন নম্বর রিটার্ন করে। উদাহরণস্বরূপ:

```
>>> mylist = list(range(0, 500, 50))
>>> print(mylist)
[0, 50, 100, 150, 200, 250, 300, 350, 400, 450]
>>> print(sum(mylist))
2250
```

পরিশিষ্ট C

পাইথনের কিছু মডিউল

সাজানো বা বাছাই বা sorts করার পাইথনে অনেক মডিউল রয়েছে। তুমি যদি সবগুলো সম্পর্কে জানতে চাও তবে docs.python.org/modindex.html এই ঠিকানায় গিয়ে পাইথনের ডকুমেন্টেশন দেখতে পারো। আমরা এখানে বেশি ব্যবহৃত কয়েকটি মডিউল সম্পর্কে আলোচনা করবো। তবে তোমার জন্য একটি সতর্কবার্তা, তুমি যদি পাইথনের ডকুমেন্টেশন দেখো, তবে সেখানে তুমি মডিউলের একটি দীর্ঘ তালিকা দেখতে পাবে যার কিছু আসলেই জটিল।

The 'random' module

তুমি যদি কখনও এমন খেলা খেলে থাকো যেখানে কাউকে ১ থেকে ১০০ পর্যন্ত সংখ্যার মাঝে একটি সংখ্যা অনুমান করতে বলবে, তাহলে তুমি জানবে র‍্যানডম মডিউল কি করতে পারে। র‍্যানডম ফাংশনের একটি সংখ্যাকে ধারণ করে যা পরবর্তী র‍্যানডম নম্বরের জন্য প্রয়োজনীয়। এটা অনেকটা কম্পিউটারকে কোন একটি সংখ্যাকে তুলে নেওয়া কথা বলার মতো। র‍্যানডম মডিউল ফাংশনের নম্বরকে ধারণ করে, কিন্তু এটি বেশি উপকারি রেনডিন্ট (randint), চয়েস (choice) এবং শাফল (shuffle) করার কাজের জন্য। প্রথম ফাংশন র‍্যানডম প্রথম ও শেষ নম্বর থেকে একটি র‍্যানডম নম্বরকে তুলে আনে (কথায় বলা যেতে পারে-১ এবং ১০০, ১০০ এবং ১০০০, ১০০০ এবং ৫০০০ এর মাঝে এবং আরও অনেক)। উদাহরণস্বরূপ :

```
>>> import random
>>> print(random.randint(1, 100))
58
>>> print(random.randint(100, 1000))
861
>>> print(random.randint(1000, 5000))
3795
```

আমরা এটি ব্যবহার করে while loop-এর সাহায্যে একটি অনুমানের খেলা খেলতে পারি:

```
import random
import sys
num = random.randint(1, 100)
while True:
    print('Guess a number between 1 and 100')
    chk = sys.stdin.readline()
    i = int(chk)
    if i == num:
        print('You guessed right')
        break
    elif i < num:
        print('Try higher')
    elif i > num:
        print('Try lower')
```

তোমার যদি একটি তালিকা থাকে এবং তুমি যদি তালিকা থেকে একটি র‍্যানডম আইটেম তুলতে চাও তবে choice ব্যবহার করতে পারো। উদাহরণস্বরূপ:

```
>>> import random
>>> list1 = [ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' ]
>>> print(random.choice(list1))
c
>>> list2 = [ 'ice cream', 'pancakes', 'trifle', 'pavlova',
'sponge' ]
>>> print(random.choice(list2))
trifle
```

এবং সর্বশেষে তুমি যদি একটি মিক্সআপ তালিকা তৈরি করতে চাও তবে shuffle ব্যবহার করো (যেমন কার্ড শাফল করা):

```
>>> import random
>>> list1 = [ 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h' ]
>>> list2 = [ 'ice cream', 'pancakes', 'trifle', 'pavlova',
'sponge' ]
>>> random.shuffle(list1)
>>> print(list1)
['h', 'e', 'a', 'b', 'c', 'g', 'f', 'd']
>>> random.shuffle(list2)
>>> print(list2)
['pancakes', 'ice cream', 'sponge', 'trifle', 'pavlova']
```

The 'sys' module

sys মডিউল গুরুত্বপূর্ণ ফাংশন system-কে ধারণ করে। পাইথনে এটি খুব দরকারী ফাংশন। sys এ কিছু উপকারী ফাংশন এবং মান হল: exit, stdin, stdout, and version।

পাইথনকে কনসোলে থামিয়ে দেয়ার জন্য exit ফাংশন আরেকটি উপায়। উদাহরণস্বরূপ তুমি যদি টাইপ কর:

```
>>> import sys
>>> sys.exit()
```

তাহলে পাইথন কনসোল থেমে যাবে। তুমি কোন সিস্টেম অর্থাৎ উইন্ডোজ, ম্যাক কিংবা লিনাক্স ব্যবহার করছো তার উপর নির্ভর করে বিভিন্ন কিছু ঘটতে পারে- কিন্তু শেষ ফল হবে এটাই যে পাইথন কনসোল রান করা বন্ধ করে দিবে।

Stdin ব্যবহার করা হয়েছে এই বইয়ের অনেক স্থানে (ষষ্ঠ অধ্যায় দেখ), এটি কাউকে সাহায্য করে একটি প্রোগ্রাম ব্যবহার করে কোন মান প্রবেশ করাতে, উদাহরণস্বরূপ :

```
>>> import sys
>>> myvar = sys.stdin.readline()
this is a test value
>>> print(myvar)
this is a test value
```

কনসোলে মেসেজ লেখার বিপরীত পদ্ধতি হলো Stdout। কিছু কারণে এটিকে প্রিন্টের মতোও বলা হয়, কিন্তু এটি কাজ করে অনেকটা ফাইলের মতো। তাই মাঝে মাঝে stdout, print এর চেয়ে বেশি উপকারী।

```
>>> import sys
>>> l = sys.stdout.write('this is a test')
this is a test>>>
```

এখানে >>> prompt আবার কেন দেখা যাচ্ছে? এটি কোন এরর না, তাই এটি মেসেজের শেষে থাকে। কারণ তুমি যখন write-কে কল করো এটি স্বয়ংক্রিয়ভাবেই পরবর্তী লাইনে চলে আসে না। write ব্যবহার করার জন্য আমাদের যা করতে হবে তাহলো:

```
>>> import sys
>>> l = sys.stdout.write('this is a test\n')
this is a test
>>>
```

(মনে রাখতে হবে যে, stdout.write ক্যারেকটারের নম্বরকে রিটার্ন করে)।

\n হল নতুন লাইনের এসকেপ ক্যারেক্টার (যা তুমি enter কী চেপে পেতে)। এসকেপ ক্যারেক্টার হলো একটি বিশেষ ক্যারেক্টার যা তুমি স্ট্রিং হিসেবে ব্যবহার করতে পারো যখন তুমি এটি সরাসরি টাইপ করতে পারো না। উদাহরণস্বরূপ, তুমি যদি মাঝখানে একটি নতুন লাইনের স্ট্রিং তৈরি করতে চাও এবং তার জন্য এন্টার বাটন চাপো তবে তুমি একটি এরর দেখতে পারবে:

```
>>> s = 'test test
File "<stdin>", line 1
s = 'test test
^
SyntaxError: EOL while scanning single-quoted string
```

তুমি এর পরিবর্তে নিউলাইন এস্কেপ ক্যারেক্টার ব্যবহার করতে পারো:

```
>>> s = 'test test\ntest'
```

সবশেষে version হল একটি উপায় যা পাইথন রান করার সময় পাইথনের ভার্সন দেখানোর ব্যবস্থা করে:

```
>>> import sys
>>> print(sys.version)
2.5.1c1 (release25-maint, Apr 12 2007, 21:00:25)
[GCC 4.1.2 (Ubuntu 4.1.2-0ubuntu4)]
```

The 'time' module

পাইথনের টাইম মডিউলটি ডিসপ্লে ফাংশনটি ধারণ করে... অবশ্যই তা সময়কে ডিসপ্লে করে। তুমি যদি টাইম ফাংশনটি কল করো তবে ফলাফল হিসাবে তুমি যা আশা করো তা নাও প্রকাশ করতে পারে:

```
>>> import time
>>> print(time.time())
1179918604.34
```

time() ফাংশন দ্বারা সত্যিকার অর্থে 1st January 1970 (00:00:00am সঠিকভাবে) থেকে সেকেন্ডসহ রিটার্ন করে। তুমি হয়তো চিন্তাও করতে পারবে না এটা কতো দরকারী। এটি ব্যবহারের সুনির্দিষ্ট উদ্দেশ্য থাকে। উদাহরণস্বরূপ,

তুমি যদি একটি প্রোগ্রাম তৈরি করো এবং জানতে চেষ্টা করো, এটি কত দ্রুত রান করে, তবে তুমি শুরুর টাইম এবং শেষ করার টাইম রেকর্ড করে রাখতে পারো এবং তুমি তার মানের তুলনা করতে পারো। উদাহরণস্বরূপ, ০ থেকে ১০০,০০০ প্রিন্ট করতে কত সময় লাগে? তুমি এটি বের করতে সহজেই একটি ফাংশন তৈরি করতে পারো:

```
>>> def lots_of_numbers(max):
... for x in range(0, max):
... print(x)
```

অতঃপর ফাংশন কল করতে পারো:

```
>>> lots_of_numbers(100000)
```

কিন্তু আমরা যদি জানতে চাই এটা কত সময় নিয়েছিল, তবে আমরা ফাংশনটি পরিবর্তন করে টাইম মডিউল ব্যবহার করতে পারি:

```
>>> def lots_of_number(max):
...         t1 = time.time()
...         for x in range(0, max):
...             print(x)
...         t2 = time.time()
...         print('it took %s seconds' % (t2-t1))
```

এবং আমরা পুনরায় এটিকে কল করতে পারি:

```
>>> lots_of_numbers(100000)
0
1
2
3
.
.
.
99997
99998
99999
it took 6.92557406425 seconds
```

এটা কিভাবে কাজ করে? প্রথমবার আমরা time() ফাংশন কল করে, ভেরিয়েবল t1 এর জন্য মান সেট করেছিলাম। অতঃপর আমরা পুনরাবৃত্তি করে সব নম্বরের প্রিন্ট আউট করেছিলাম। পুনরায় আমরা time() ফাংশন কল করেছিলাম এবং এই সময়ে ভেরিয়েবলের মান হিসাবে t2 নির্ধারণ করেছিলাম। এটা করতে মাত্র কয়েক সেকেন্ড লেগেছিল। এখানে t2 এর মান বেশি t1 (সেকেন্ডের নাম্বার 1st Jan 1970 থেকে শুরু হয়ে বৃদ্ধি পাচ্ছিল) থেকে বেশি ছিল, তাই আমরা যদি t1 থেকে t2 বাদ দেই তাহলে আমাদের হাতে থাকবে ঐ সকল নম্বর প্রিন্ট করা নম্বর।

Time মডিউলে আরো কয়েকটি ফাংশন যুক্ত করা আছে: asctime, ctime, local-time, sleep, strftime, and strptime.

asctime একটি টাপল হিসেবে তারিখকে নিয়ে নেয় (মনে রাখতে হবে একটি টাপল হলো তালিকা যার মান পরিবর্তন করা যায় না) এবং এটি একটি পঠণযোগ্য ফর্মে রুপান্তর করে। কোন আর্গুমেন্ট ছাড়াই এটিকে কল করা

যায় এবং এটি বর্তমান সময় এবং তারিখ পঠণযোগ্য ফর্মে প্রদর্শন করে।

```
>>> import time
>>> print(time.asctime())
Sun May 27 20:11:12 2007
```

একটি আর্গুমেন্টসহ এটিকে কল করলে, প্রথমে আমাদের তারিখ এবং সময়ের জন্য সঠিক মানসহ একটি টাপল তৈরি করতে হবে। চলো আমরা একটি টাপল অ্যাসাইন করি t ভেরিয়েবলের মধ্যে।

```
>>> t = (2007, 5, 27, 10, 30, 48, 6, 0, 0)
```

এখানে মানের ক্রমান্বয় হচ্ছে বছর, মাস, দিন, ঘন্টা, মিনিট, সেকেন্ড, সপ্তাহের দিন (০ হলো সোমবার, ১ হলো মঙ্গলবার এভাবে রবিবার পর্যন্ত যার মান ৬) সর্বশেষে বছরের দিন এবং এটি ডেলাইট সেভিং কিনা তার মান (ডেলাইট সেভিং হলে ১, না হলে ০)। টাপলে যদি আমরা asctime কল করি তাহলে পাবো:

```
>>> import time
>>> t = (2007, 5, 27, 10, 30, 48, 6, 0, 0)
>>> print(time.asctime(t))
Sun May 27 10:30:48 2007
```

কিন্তু টাপলে মান রাখার সময় খুব সাবধান থাকতে হবে। তুমি যদি একটি ভুল মান রাখো তবে অর্থবিহীন তারিখ দেখাবে:

```
>>> import time
>>>  t = (2007, 5, 27, 10, 30, 48, 0, 0, 0)
>>> print(time.asctime(t))
Mon May 27 10:30:48 2007
```

কারণ 'day of the week' হিসাবে নির্ধারণ করা হয়েছিল ০ (৬ এর পরিবর্তে), তাই asctime চিন্তা করে যে মে মাসের ২৭ তারিখ সোমবার, কিন্তু এটা আসলে ছিল রবিবার। ctime ফাংশন একটি নাম্বারকে সেকেন্ডে পরিবর্তন করার জন্য ব্যবহার করা হয়। উদাহরণস্বরূপ, আমরা এই অংশের শুরুটা ব্যাখ্যা করতে time() ফাংশনটি ব্যবহার করেছিলাম:

```
>>> import time
>>> t = time.time()
>>> print(t)
1180264952.57
>>> print(time.ctime(t))
Sun May 27 23:22:32 2007
```

ফাংশনটি আঞ্চলিক সময় হিসেবে তারিখ এবং সময়কে রিটার্ন করে একটি টাপল হিসেবে এবং যে মান আমরা ব্যবহার করেছিলাম সেই ধারাবাহিকতায় তা প্রদর্শন করে:

```
>>> import time
>>> print(time.localtime())
(2007, 5, 27, 23, 25, 47, 6, 147, 0)
```

এই মানকে আমরা asctime হিসেবে পাস করতে পারি:

```
>>> import time
>>> t = time.localtime()
>>> print(time.asctime(t))
Sun May 27 23:27:22 2007
```

তুমি যখন একটি প্রোগ্রামকে একটি নির্দিষ্ট সময় দেরি করে রান করাতে চাও তখন sleep ফাংশনটি খুবই কাজে লাগে। উদাহরণস্বরূপ, তুমি যদি প্রতি সেকেন্ডে একটি করে নম্বর প্রিন্ট করতে চাও তাহলে এই loop ব্যবহার করে খুব বেশি সফল হতে পারবে না:

```
>>> for x in range(1, 61):
...
print(x)
...
1
2
3
4
```

এটি সাথে সাথেই ১ থেকে ৬০ পর্যন্ত প্রিন্ট করবে। তুমি যদি পাইথনকে বল যে প্রতিটি প্রিন্ট স্টেটমেন্টের আগে এক সেকেন্ডের জন্য বন্ধ হতে তাহলে:

```
>>> for x in range(1, 61):
...
print(x)
...
time.sleep(1)
...
```

তাহলে প্রতিটি নম্বরের প্রদর্শনের মাঝে একটি সংক্ষিপ্ত বিরতি হবে (প্রতিবার এক সেকেন্ড)। একটি কম্পিউটার যদি আমরা বন্ধ হতে বলি তবে সবসময় দরকারী নাও হতে পারে। কিন্তু যদি সময় বলে দেই তবে তা দরকারী হতে পারে। তুমি তোমার এলার্ম ঘড়ির কথা মনে করতে পারো যা তোমাকে প্রতিদিন সকালে জাগিয়ে দেয়। তুমি যখন স্লিপ বাটনে চাপ দাও তখন এটি কিছু সময়ের জন্য থেমে যায়। তোমাকে ঘুমানোর জন্য কিছু সময় দেয় (যতক্ষণ তোমাকে সকালের নাস্তার জন্য কেউ না ডাক দেয়)। sleep ফাংশন এক্ষেত্রে তাই প্রয়োজনীয়।

strftime ফাংশন তারিখ এবং সময়ের প্রদর্শিত মানের পরিবর্তনের উপায় হিসাবে ব্যবহার করা হয় এবং strptime একটি স্ট্রিং নিতে ও তারিখ/সময় টাপলে পরিবর্তন করার কাজে ব্যবহার করা হয়। চলো আমরা প্রথমে strftime দেখি, তবে তার আগে দেখতে হবে কিভাবে asctime ব্যবহার করে টাপলকে স্ট্রিংয়ে পরিবর্তন করা হয়:

```
>>> t = (2007, 5, 27, 10, 30, 48, 6, 0, 0)
>>> print(time.asctime(t))
Sun May 27 10:30:48 2007
```

এটি প্রায় ক্ষেত্রেই খুব ভাল কাজ করে। কিন্তু কি হবে যখন তুমি যেভাবে স্ট্রিং প্রদর্শিত হয় সেভাবে স্ট্রিং প্রর্দশন করতে চাইবে না। কি হবে তুমি যদি শুধু মাত্র তারিখ এবং সময় প্রদর্শন করতে চাও? আমরা এটি করতে পারি strftime ফাংশন দ্বারা:

```
>>> print(time.strftime('%d %b %Y', t))
27 May 2007
```

তুমি দেখতে পারবে, strftime দুইটি আর্গুমেন্ট নিতে পারে: প্রথমটি date/time ফরমেটে (যেটা বর্ণনা করবে কিভাবে তারিখ ও সময় প্রদর্শন করবে) এবং দ্বিতীয়টি টাপল যা time-এর মান ধারণ করে। %d %b %Y ফরমেট দ্বারা বোঝা যায়, প্রথমে দিন, পরে মাস এবং শেষে বছর প্রদর্শন করবে। আমরা মাসকে একটি নম্বর হিসাবেও দেখাতে পারি, উদাহরণস্বরূপ:

```
>>> print(time.strftime('%d/%m/%Y', t))
27/05/2007
```

এই ফরমেট বলে যে, প্রথমে দিন দেখাবে, তারপর একটি forward-slash দেখাবে, অতঃপর মাসকে একটি নাম্বার হিসাবে দেখাবে, তারপর আবারও forward-slash এবং পরে বছর দেখাবে। প্রচুর সংখ্যক ভিন্ন মান আছে যা আমরা একটি ফরমেটে প্রদর্শন করতে পারি:

%a	সপ্তাহের দিনগুলো প্রদর্শনের একটি সংক্ষিপ্ত ভার্সন (উদাহরণস্বরূপ- Mon, Tues, Wed, Thurs, Fri, Sat, and Sun)
%A	সপ্তাহের পুরো নাম (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)
%b	মাসের সংক্ষিপ্ত নামের ভার্সন (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct , Nov, Dec)
%B	মাসের পুরো নাম (January, February, March, April, May, and so on)
%c	পুরো তারিখ ও সময়, asctime এ ব্যবহারের মতোন ফরমেট হিসাবে ব্যবহার করা হয়।
%d	নম্বর হিসাবে মাসের একটি দিন (০১ থেকে ৩১ পর্যন্ত)
%H	দিনের একটি ঘন্টা, ২৪ ঘন্টা ফরমেট (০০ থেকে ২৩ পর্যন্ত)
%I	দিনের একটি ঘন্টা, ২১ ঘন্টা ফরমেট (০১ থেকে ১২ পর্যন্ত)
%j	নম্বর হিসাবে বছরের একটি দিন (০০১ থেকে ৩৬৬ পর্যন্ত)
%m	নম্বর হিসাবে মাস (০১ থেকে ১২ পর্যন্ত)
%M	নম্বর হিসাবে মিনিট (০০ থেকে ৫৯ পর্যন্ত)
%p	সকাল অথবা দুপুর AM অথবা PM
%S	নম্বর হিসাবে সেকেন্ড
%U	নম্বর হিসাবে একটি বছরের সপ্তাহের নাম (০০ থেকে ৫৩ পর্যন্ত)
%w	সপ্তাহের একটি দিন নাম্বার হিসাবে। রবিবার ০, সোমবার ১, শনিবার পর্যন্ত যার মান ৬
%x	সাধারণ একটি তারিখ ফরমেট (সাধারণত month/day/year- উদাহরণস্বরূপ- ০৩/২৫/০৭)
%X	সময়ের একটি সাধারণ ফরমেট (সাধারণত hour:minutes:seconds- উদাহরণস্বরূপ- 10:30:53)
%y	বছর ২ ডিজিট হিসাবে (উদাহরণস্বরূপ- ২০০৭ হতে পারে ০৭)
%Y	বছর ৪ ডিজিটের (যেমন- ২০০৭)

strptime ফাংশন strftime ফাংশনের ঠিক উল্টো। strftime একটি স্ট্রিং-কে টাপল হিসেবে রূপান্তর করে যা ধারণ করে তারিখ এবং সময়ের মান। এটি সবসময় ফরমেট স্ট্রিংয়ের একই মানকে গ্রহণ করে। ফাংশনটি ব্যবহার করে

একটি উদাহরণ:

```
>>> import time
>>> t = time.strptime('05 Jun 2007', '%d %b %Y')
>>> print(t)
(2007, 6, 5, 0, 0, 0, 1, 156, -1)
```

যদি আমাদের স্ট্রিংয়ে তারিখ হয় day/month/year (উদাহরণ হিসাবে- 01/02/2007), আমরা ব্যবহার করতে পারি:

```
>>> import time
>>> t = time.strptime('01/02/2007', '%d/%m/%Y')
>>> print(t)
(2007, 2, 1, 0, 0, 0, 3, 32, -1)
```

অথবা তারিখ যদি হয় month/day/year, তাহলে আমরা ব্যবহার করতে পারি:

```
>>> import time
>>> t = time.strptime('03/05/2007', '%m/%d/%Y')
>>> print(t)
(2007, 3, 5, 0, 0, 0, 0, 64, -1)
```

আমরা যৌথভাবে দুইটি ফাংশন ব্যবহার করতে পারি, একটি স্ট্রিং কে অন্য আরেকটি ফরমেটে পরিবর্তন করার কাজে, চলো আমরা একটি ফাংশন হিসাবে কাজটি করি:

```
>>> import time
>>> def convert_date(datestring, format1, format2):
... t = time.strptime(datestring, format1)
... return time.strftime(format2, t)
...
```

আমরা এই ফাংশন ব্যবহার করতে পারি তারিখ স্ট্রিং এই ফরমেটে পরিবর্তন করতে এবং তারপর আমরা যে ফরমেটে তারিখ কে রিটার্ন করতে চাই তা হলো:

```
>>> print(convert_date('03/05/2007', '%m/%d/%Y', '%d %B %Y'))
05 March 2007
```

পরিশিষ্ট D

"এসো নিজে করি " এর উত্তর

আমরা প্রতি অধ্যায়ে যে সমস্ত প্রশ্ন করেছিলাম তার এই উত্তর এই "Things to try" সেকশনে পাবে।

দ্বিতীয় অধ্যায়

১. অনুশীলনী ১ এর উত্তর সম্ভবত নিচের মতো হবে:

```
>>> toys = [ 'car', 'Nintendo Wii', 'computer', 'bike' ]
>>> foods = [ 'pancakes', 'chocolate', 'ice cream' ]
>>> favourites = toys + foods
>>> print(favourites)
['car', 'Nintendo Wii', 'computer', 'bike', 'pancakes',
'chocolate', 'ice cream']
```

২. অনুশীলনী ২ এর উত্তর খুবই সাধারণভাবে ৩ কে ২৫ দিয়ে গুণ করে এবং ১০ এর সাথে ৩২ গুণ করে যোগ করা। নিচের ইকুয়েশনটি ঐ ইকুয়েশনের রেজাল্ট দেখায়:

```
>>> print(3 * 25 + 10 * 32)
395
```

যা হোক, আমরা দ্বিতীয় অধ্যায়ে ব্র্যাকেটের ব্যবহার দেখেছি, তুমি নিজেই চিন্তা করতে পারো তোমার ইকুয়েশনে কোথায় কোথায় ব্র্যাকেট বসাতে চাও। তুমি এটা এভাবেও করতে পারো:

```
>>> print((3 * 25) + (10 * 32))
395
```

দুইটার উত্তর একই, কারণ যোগের আগেই গুণ করেছে, এই সমীকরণে, দুইটি গুণের কাজ প্রথমে করা হয়েছে এবং রেজাল্ট যোগ করা হয়েছে। যা হোক, দ্বিতীয় সমীকরণটি প্রথমটি থেকে কিছুটা ভালো। কারণ এটা পাঠকের জন্য সুস্পষ্ট যে কোন অপারেশনটি প্রথমে কাজ করবে। একজন কম জানা প্রোগ্রামারও (যে অপারেটরের ক্রমান্বয় জানে না) প্রথম সমীকরণের বেলায় বুঝতে পারবে তুমি ৩ এর সাথে ২৫ গুণ করে তার সাথে ১০ যোগ করতে চাচ্ছ। এর পর এই ফলাফলের সাথে ১০ যোগ করে সেই ফলাফলের সাথে ৩২ গুণ করতে চাচ্ছ (তাহলে রেজাল্ট হয় ২৭২০- যা সম্পূর্ণভাবেই ভুল)। ব্র্যাকেটসহ বোঝা যায় কোন হিসাবটি আগে করতে হবে।

৩. তিন নম্বর উদাহরণের উত্তর নিচের মতো হবে:

```
>>> first_name = 'Maria'
>>> second_name = 'Ahamed'
>>> print('My name is %s %s' % (first_name, second_name))
My name is Mary Ahamed
```

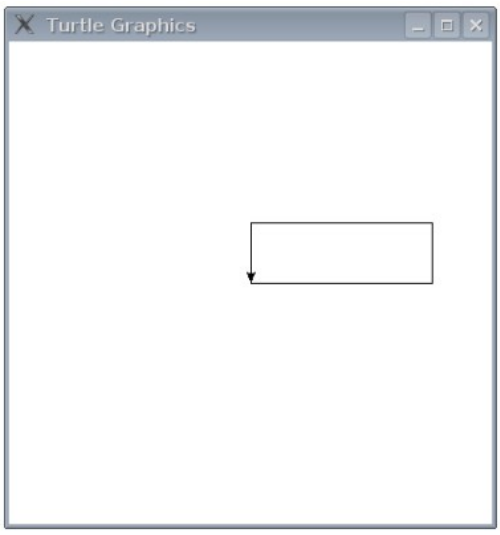
তৃতীয় অধ্যায়

১। একটি চতুর্ভূজ বর্গাকৃতির মতো, এর দুই বাহু অপর দুই বাহুর চেয়ে বড়। আমরা যদি টার্টলকে অপারেটর ব্যবহার করে এটি করতে বলি তাহলে সহজেই তুমি একটি চতুর্ভূজ আঁকতে পারবে:

- একটি নির্দিষ্ট নাম্বার পিক্সেল পর্যন্ত এগিয়ে যাও
- বামে ঘুরো
- একটি কম পিক্সেল পর্যন্ত সামনে আগাও
- বামে ঘুরো
- প্রথমবার যত পিক্সেল সামনে গিয়েছিলে তত পিক্সেল এগিয়ে যাও
- বামে ঘুরো
- দ্বিতীয় বার যত পিক্সেল সামনে গিয়েছিলে তত পিক্সেল সামনে যাও

উদাহরণস্বরূপ, চিত্র D.1 এর চতুর্ভূজ আঁকতে নিচের কোডটি ব্যবহার করা হয়েছিল:

```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(150)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(150)
>>> t.left(90)
>>> t.forward(50)
```



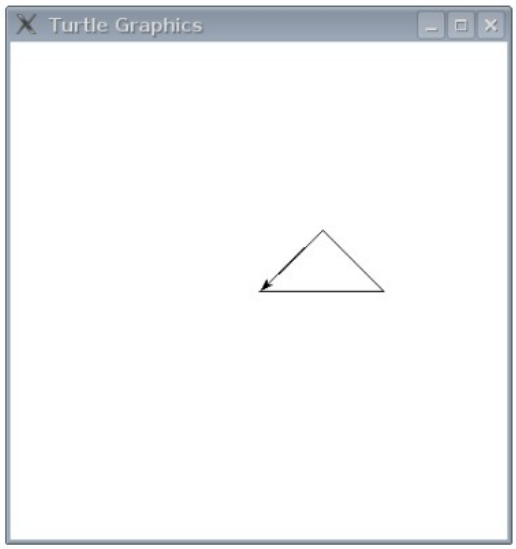
চিত্র ডি-১: টার্টল একটি আয়তক্ষেত্র আঁকছে

২। একটি চতুর্ভূজ আঁকা কিছুটা জটিল, কারণ তোমাকে এর জন্য কোণের মাপ এবং দৈর্ঘ্য সম্পর্কে জানতে হবে। তুমি যদি বিদ্যালয়ে কোণ সম্পর্কে কিছু না পড়ে থাকো তবে তুমি যা আশা করছো তা করাটা কঠিন হতে পারে। তুমি এই কোডটি ব্যবহার করে একটি সাধারণ ত্রিভুজ আঁকতে পারো (চিত্র D2 দেখ)।

```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(100)
>>> t.left(135)
>>> t.forward(70)
>>> t.left(90)
>>> t.forward(70)
```

পঞ্চম অধ্যায়

১। প্রথম বার print হওয়ার পর loop থামবে। তাই যখন তুমি তোমার পাইথন কনসোলে এই কোডটি চালাবে তুমি তখন পাবে:



চিত্র ডি-২: টার্টল একটি ত্রিভুজ আঁকছে

```
>>> for x in range(0, 20):
…          print('hello %s' % x)
…          if x < 9:
...            break
hello 0
```

তার কারণ ছিল এটা প্রথমবার প্রিন্ট হওয়ার পর থামবে এবং এটা চলবে প্রথম loop পর্যন্ত, এখানে ভেরিয়েবল x এর মান শূণ্য। যেহেতু শূণ্য ৯ এর চেয়ে ছোট, তাই ব্রেক স্টেটমেন্ট লুপকে থামাবে যে কোন সময়ে।

২। তুমি যদি ৩% লাভ পাও তবে কত টাকা পাবে তা বের করো, তোমার প্রয়োজন পড়বে মোট নাম্বারকে ০.০৩ দিয়ে গুণ করার। এর জন্য আমরা একটি ভেরিয়েবল তৈরি করতে পারি, এটাকে আমাদের সেভিংসের পরিমান জানাতে পারি:

```
>>> amount = 100
```

এক বছরের জন্য যে ইন্টারেস্ট পাওয়া যাবে তার সাথে 0.03 গুণ করতে হবে:

```
>>> amount = 100
>>> print(amount * 0.03)
3.0
```

যা মাত্র তিন টাকা! এটা মন্দ না কারণ এর জন্য আমাকে কিছু করতে হয়নি। আমাদের এই মানকে প্রিন্ট করতে হবে এবং এটাকে মোটের সাথে যোগ করতে হবে। আমরা যদি ১০ বছরের জন্য এটি করতে চাই তবে এই কাজটি দশবার করতে হবে:

```
>>> amount = 100
>>> for year in range(1, 11):
...  interest = amount * 0.03
…      print('interest earned for year %s is %s' % (year,
interest))
…      amount = amount + interest
...
interest earned for year 1 is 3.0
interest earned for year 2 is 3.09
interest earned for year 3 is 3.1827
interest earned for year 4 is 3.278181
interest earned for year 5 is 3.37652643
interest earned for year 6 is 3.4778222229
interest earned for year 7 is 3.58215688959
interest earned for year 8 is 3.68962159627
interest earned for year 9 is 3.80031024416
interest earned for year 10 is 3.91431955149
```

প্রথম লাইনে আমরা for-loop তৈরি করেছি, year ভেরিয়েবল ব্যবহার করেছি এবং ফাংশন রেঞ্জ ১ থেকে ১০ পর্যন্ত কাউন্ট করেছে। দ্বিতীয় লাইন ইন্টারেস্ট গণনা করেছে এবং ভেরিয়েবলের মানের সাথে ০.৩০ গুণ করেছে। পরবর্তী লাইনের প্রিন্ট স্টেটমেন্ট- যাতে % ব্যবহার করা হয়েছে যাতে বছরের মান এবং ইন্টারেস্ট এর মান যুক্ত করে। সর্বশেষের লাইনের, আমরা এমাউন্টে ফিরে এসে ইন্টারেস্ট যোগ করেছি। প্রিন্ট লাইনে দশমিকের পরের ঘরগুলো অনেক সময়ই বিভ্রান্ত তৈরি করে। কিন্তু তুমি বলতে পারবে প্রতি বছর তোমার যোগ করা ইন্টারেস্টে কিছু পরিমান ইন্টারেস্ট বৃদ্ধি পাচ্ছে। এই কোডটি দরকারী হবে যখন আমরা সবসময়ই প্রতি বছর কিছু সেইভ করবো:

```
>>> amount = 100
>>> for year in range(1, 11):
…      interest = amount * 0.03
…      print('interest earned for savings %s for year %s is   %s' %
…      (amount, year, interest))
…      amount = amount + interest
...
interest earned for savings 100 for year 1 is 3.0
interest earned for savings 103.0 for year 2 is 3.09
interest earned for savings 106.09 for year 3 is 3.1827
interest earned for savings 109.2727 for year 4 is 3.278181
interest earned for savings 112.550881 for year 5 is 3.37652643
interest earned for savings 115.92740743 for year 6 is 3.4778222229
interest earned for savings 119.405229653 for year 7 is 3.58215688959
interest earned for savings 122.987386542 for year 8 is 3.68962159627
interest earned for savings 126.677008139 for year 9 is 3.80031024416
interest earned for savings 130.477318383 for year 10 is 3.91431955149
```

একটি for-loop কে সহজেই ফাংশনে রূপান্তর করা যায়। ফাংশনটি দেখতে এই রকম হবে:

```
>>> def calculate_interest(amount, rate):
...   for year in range(1, 11):
...         interest = amount * rate
...         print('interest earned for savings %s for year %s is %s' %
...               (amount, year, interest))
...         amount = amount + interest
```

যদি তুমি এই কোডের সাথে তুলনা করো, তুমি দেখতে পারবে যে, প্রথম লাইনের অর্জিনাল কোডে একটি মাত্র পরিবর্তন করা হয়েছে (০.০৩ হলো প্যারামিটার রেট)। কারণ amount আগে থেকেই একটি ভেরিয়েবল, যখন এটি প্যারামিটার করা হবে তখন কোন পরিবর্তন করা যাবেনা। যখন তুমি এই ফাংশনটি রান করাবে তখন সবসময় একই আউটপুট পাবে:

```
>>> calculate_interest(100, 0.03)
interest earned for savings 100 for year 1 is 3.0
interest earned for savings 103.0 for year 2 is 3.09
interest earned for savings 106.09 for year 3 is 3.1827
interest earned for savings 109.2727 for year 4 is 3.278181
interest earned for savings 112.550881 for year 5 is 3.37652643
interest earned for savings 115.92740743 for year 6 is 3.4778222229
interest earned for savings 119.405229653 for year 7 is 3.58215688959
interest earned for savings 122.987386542 for year 8 is 3.68962159627
interest earned for savings 126.677008139 for year 9 is 3.80031024416
interest earned for savings 130.477318383 for year 10 is 3.91431955149
```

২। ইয়ারকে pass ফাংশনের পরিবর্তন করায় ইয়ার প্যারামিটারে খুব অল্পই পরিবর্তন করা যাবে:

```
>>> def calculate_interest(amount, rate, years):
...   for year in range(1, years):
...         interest = amount * rate
...         print('interest earned for savings %s for year %s is %s' %
...               (amount, year, interest))
...         amount = amount + interest
```

আমরা এখন সহজেই প্রথম এমাউন্ট, ইন্টারেস্টের হার, বছরের সংখ্যা পরিবর্তন করতে পারি:

```
>>> calculate_interest(1000, 0.05, 6)
interest earned for savings 1000 for year 1 is 50.0
interest earned for savings 1050.0 for year 2 is 52.5
interest earned for savings 1102.5 for year 3 is 55.125
interest earned for savings 1157.625 for year 4 is 57.88125
interest earned for savings 1215.50625 for year 5 is 60.7753125
```

৩। আমরা যে ফাংশন তৈরি করেছি তাতে একটি ছোট প্রোগ্রাম করা অনেক জটিল। প্রথমে আমাদের sys মডিউল ইমপোর্ট করতে হবে, যাতে আমরা ইনপুট নিতে বলতে পারি। তারপর আমাদের প্রতিটি মানের জন্য ব্যবহারকারীকে প্রমোট করতে হবে। এখন থেকে পৃথকভাবে ফাংশন একই সাথে বিশৃঙ্খলভাবে থাকবে:

```
>>> import sys
>>> def calculate_interest():
...   print('Enter the amount you have to save')
...   amount = float(sys.stdin.readline())
...    print('Enter the interest rate')
...    rate = float(sys.stdin.readline())
...   print('Enter the number of years')
...   years = int(sys.stdin.readline())
...    for year in range(1, years):
...          interest = amount * rate
...          print('interest earned for savings %s for year %s is %s' %
...               (amount, year, interest))
...         amount = amount + interest
```

আমরা যখন ফাংশনটি রান করাবো তখন নিচের মতো দেখতে পাবো:

```
>>> calculate_interest()
Enter the amount you have to save
500
Enter the interest rate
0.06
Enter the number of years
12
interest earned for savings 500.0 for year 1 is 30.0
interest earned for savings 530.0 for year 2 is 31.8
interest earned for savings 561.8 for year 3 is 33.708
interest earned for savings 595.508 for year 4 is 35.73048
interest earned for savings 631.23848 for year 5 is 37.8743088
interest earned for savings 669.1127888 for year 6 is 40.146767328
interest earned for savings 709.259556128 for year 7 is 42.5555733677
interest earned for savings 751.815129496 for year 8 is 45.1089077697
interest earned for savings 796.924037265 for year 9 is 47.8154422359
interest earned for savings 844.739479501 for year 10 is 50.6843687701
interest earned for savings 895.423848271 for year 11 is 53.7254308963
```

অষ্টম অধ্যায়

১। অক্টাগণ তৈরির জন্য একটি কঠিন পন্থা আছে, এবং একটি সহজ রাস্তা আছে। কঠিন উপায় আসলে কঠিন না তবে জটিল, এটা জটিল কারণ এর জন্য অনেক বেশি টাইপ করতে হবে:

```
import turtle
t = turtle.Pen()
>>> t.forward(50)
>>> t.right(45)
>>> t.forward(50)
>>> t.right(45)
>>> t.forward(50)
>>> t.right(45)
>>> t.forward(50)
>>> t.right(45)
>>> t.forward(50)
>>> t.right(45)
```

```
>>> t.forward(50)
>>> t.right(45)
>>> t.forward(50)
>>> t.right(45)
>>> t.forward(50)
```

তুমি দেখতে পারো যে এই কোডে আমরা টার্টলকে ৫০ পিক্সেল সামনে যেতে বলেছি, অতঃপর ডানদিকে ৪৫ ডিগ্রি ঘুরতে বলেছি। এটি আমরা আট বার করতে বলেছি। যার জন্য প্রচুর সময় প্রয়োজন। একটি অক্টাগন তৈরির সহজ উপায় হলো নিচের কোড (যা চিত্র D3 এর মতো অক্টাগন তৈরি করে):

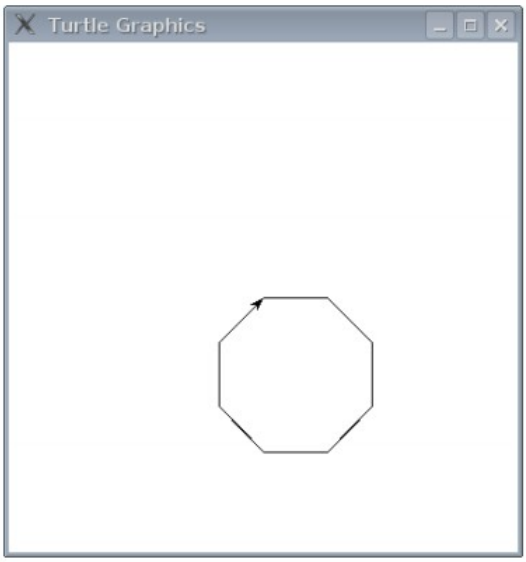
```
>>> for x in range(0,8):
...   t.forward(50)
...    t.right(45)
```

২। তুমি যদি অষ্টম অধ্যায়ে অন্য আরেকটি ফাংশন দেখে থাকো, তাহলে জেনে থাকবে কিভাবে একটি ফিলড শেপ তৈরি করা হয়। আমরা অক্টাগন ফাংশনকে একটি ফাংশনে পরিনত করে তাতে রং দিতে পারি, কিন্তু আমরা সর্বদা হেক্সাকালার ফাংশন ব্যবহার করতে চাইবো।

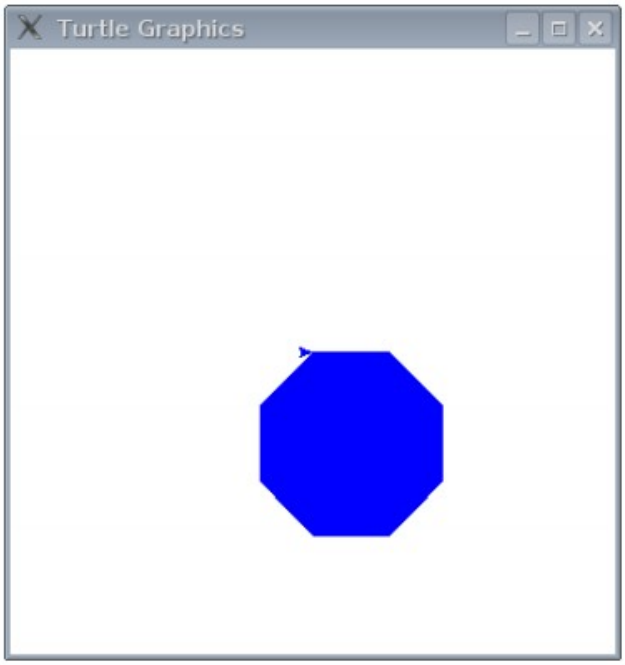
```
>>> def octagon(red, green, blue):
...   t.color(red, green, blue)
...   t.begin_fill()
...    for x in range(0,8):
...        t.forward(50)
...        t.right(45)
...    t.end_fill()
```

আমরা কালার সেট করেছি, অতঃপর এটি পূর্ণ করেছি। তারপর আমরা অক্টাগন আঁকতে for-loop রান করিয়েছি। সর্বশেষে আমরা মূল জায়গায় ফিরিয়ে এনে শেপটাকে ভর্তি করছি। নিচে নীল অক্টাগনটি দেখ (চিত্র D.4 দেখ):

```
>>> octagon(0, 0, 1)
```



চিত্র ডি-৩: টার্টল একটি অষ্টভুজ আঁকছে



চিত্র ডি-৪: টার্টল একটি নীল অষ্টভুজ আঁকছে

নির্ঘণ্ট (Index)

এডিশন (addition)
ব্লকস অব কোড (blocks of code)
কনডিশনস (conditions)
কমবাইনিং (combining)
ডেট/টাইম ফরমেট (date/time formats,)
ডিগ্রি (degrees)
ডিভিশন (division)
ইকুয়ালিটি (equality)
এসকেপ ক্যারেকটরস (escape characters)
ফ্লোটিং পয়েন্ট (floating point)
ফর-লুপ (for-loop)
ফাংশন (functions)
এবিএস (abs)
বোল (bool)
সিএমপি (cmp)
ডির (dir)
ইভাল (eval)
ফাইল (file)
ক্লোজ (close)
রিড (read)
হোয়াইট (write)
ফ্লোট (float)
ইন্ট (int)
লেন (len)
ম্যাক্স (max)
মিন (min)
রেঞ্জ range)
সাম (sum)
হেক্সাডিসিমেল কালারস (hexadecimal colors)
ইফ-স্টেটমেন্ট (if-statement)
ইফ-দেন-ইলসি-স্টেটমেন্ট (if-then-else-statement)
কীওয়ার্ড (keywords)
এন্ড (and)
অ্যাজ (as)
অ্যাজার্ট (assert)
ব্রেক (break)
ক্লাশ (class)
ডেল (del)
মডিউলস (modules)
অস (os)
র‍্যানডম (random)
চয়েস (choice)
র‍্যানডিন্ট (randint)
র‍্যানডেঞ্জ (randrange)
শাফল (shuffle)
সিস (sys)
এক্সিট (exit)
এসটিডিন (stdin)
এসটিআউট (stdout)
ভার্সন (version)
টাইম (time)
এসসিটাইম (asctimie)
সিটাইম (ctime)
লোকালটাইম (localtime)
স্লিপ (sleep)
এসটিআরটিপিটাইম (strftime)
টাইম (ফাংশন) (time (funtion))
টিংকার (tkinkter)
বেসিক এনিমেশন (basic animation)
বাইন্ড_অল (bind_all)
ক্যানভাস (Canvas)
ক্রিয়েট_এআরসি (create_arc)
ক্রিয়েট_ইমেজ (create_image)
ক্রিয়েট_লাইন (create_line)
ক্রিয়েট_ওভাল (create_oval)
ক্রিয়েট_পলিগন (create_polygon)
ক্রিয়েট_র‍্যাক্টঙ্গল (create_rectange)
ইভেন্ট (events)
মুভ (move)
মডুলু অপারেটর (modulo operator)
মাল্টিলাইন স্ট্রিং (multi-line string)
মাল্টিপ্লিকেশন (multiplication)
নেইমড প্যারামিটার (named parameters)
নান (none)
অপারেটর (operators)
অর্ডার অফ অপারেশনস (order of operations)
পেন (pen)

ইলইফ (elif)
ইলসি (else)
এক্সেপ্ট (except)
এক্সেক (exec)
ফাইনালি (finally)
ফর (for)
ফ্রম (from)
গ্লোবাল (global)
ইফ (if)
ইমপোর্ট (import)
ইন (in)
ইজ (is)
ল্যামডা (lambda)
নট (not)
অর (or)
পাস (pass)
রেইজ (raise)
রিটার্ন (return)
ট্রাই (try)
হোয়াইল (while)
ইয়েল্ড (yield)
লিস্ট (list)
এপেনডিং (appending)
জয়েনিং (joining)
রিমুভিং (removing)
রিপ্লেসিং (replacing)
পিক্সেল (pixels)
পাইথন (python)
পাইথন কনসোল (python console)
রিটার্ন (return)
স্কোপ (scope)
স্ট্রিং (strings)
সাবট্রাকশন (subtraction)
টাপল (tuples)
টার্টল (turtle)
এডভান্স (advanced)
ব্যাকওয়ার্ড (backword)
ক্লিয়ার (clear)
কালার (color)
ব্ল্যাক (black)
ডাউন (স্টার্ট ড্রয়িং) (down (start drawing
ফিল (fill)
ফরোয়ার্ড (forward)
রিসেট (reset)
টারনিং লেফট (turning left)
টারনিং রাইট (turning right)
আপ (স্টপ ড্রয়িং) (up (stop drawing))
ভেরিয়েবল (variable)
ভেরিয়েবলস (variables)
হুয়াইল-লুপ (while-loop)
হুয়াইট স্পেস (white-space)