Python for Absolute Beginners

Introduction to Programming

Assoc. Prof. Dr. Bunyarit Uyyanonvara

Sirindhorn International Institute of Technology (SIIT)

Thammasat University
Overview

This book is the main companion of Sirindhorn International Institute of Technology's ITS100 - Introduction to computer and programming. This edition is free and for academic use only.

This book is available for free download at http://www.siit.tu.ac.th/its100.

ITS100 is one of the biggest subjects at SIIT at about 300-400 students per semester. All students in SIIT have to pass this subject no matter they are engineering students or management students. This subject not only teaches students how to code but also get them to think systematically.

In designing this subject, we take into account that most of student has very little programming background. Many of them also have limited experience in a computer-like step-by-step procedural thinking process. Teaching this huge group of students to program is extremely difficult. That’s why we think Python programming language is the best choice to get them started with programming.

Python (http://www.python.org/about/gettingstarted/) is introduced and it is now widely accepted by many universities (such as MIT, http://tinyurl.com/dxk7bsg) as an entry point for simple programming concept. Not only the traditional lectures and labs that provide in this subject, we also added a lot of fun activities into this ITS100 curriculum. The subject is project-based learning subject with extra activities that are designed to make students love programming. They should feel that programming is fun (even though it might not be easy, but, at least, it’s fun.) The key word “Programming is Fun” will be injected into all the classes and additional activities. The approach of the course is project-based learning.

Goals

Students who passed this subject will love programming. They will get the feeling that programming is cool and fun. They will love SIIT for its teaching innovation.
### Table of Content

- Chapter 1: Introduction to programming with Python
- Chapter 2: Turtle
- Chapter 3: Variables
- Chapter 4: Predefined functions
- Chapter 5: Conditional Structure (if)
- Chapter 6: Repetition Structure (for, while)
- Chapter 7: User-defined function
- Chapter 8: List & String Manipulation
- Chapter 9: Animation & Games
Chapter 1

First-time Programmer

Learn to code is learn to think

If you choose to read this book, then I assume that you have little experience of writing a program or, as I shall say from now, coding. I know that many of you have been asking “why do we have to write a program? Why do we have to code?”. OK, that’s why this chapter is here. The chapter will get you through the basic knowledge of programming and also introduce you to a very simple yet powerful programming language, Python.

OK, if you don’t know yet why the big institutes set their curriculum so that all the students in the institutes have to pass at least one programming course before they can graduate, you will know in a few minutes. In my institute, no matter what major you are, engineering or management, you will have to start your first year with a programming subject.

Coding is simply a way that we, as a human, talk to a computer. Of course, computer doesn’t understand human language directly, we need to talk (or rather to order) to it by some specific language. Learning how to code is to learn how to talk to the most stupid thing in the world, and that’s the computer. Computer doesn’t have brain like we do, it
follows our instructions exactly. You have to order it command by command. If you give the wrong instructions, it will not do what you expected or even clash sometimes. So to make it work for what you wanted, you have to talk to computer with very clear step-by-step commands. You have to deliver your thought, your idea, to the most stupid thing. That process is called coding.

If you can succeed in explaining the most stupid thing about your idea, about what you are thinking, so you learn how to think systematically and you will then have no problem expressing your idea or thought to someone out of your field. Learn to code is to learn to think. That’s said by Steve Jobs. But people learn to code for different reasons, some learn for fun, some learn for money. As you may have heard about many young students like Mark Zuckerberg from Facebook, or the talented Larry Page and Sergey Brin, the cofounder of Google making their fortune out of coding.

Before we go further, I want to recommend you to watch the VDO on the first page of code.org (or access it on Youtube at https://www.youtube.com/watch?v=nKlu9yen5nc) It’s just 5 minutes and you will see those great people think about coding.

It’s great and inspiring, isn’t it? You know what? those great people just started their big business like you are doing now, they learn how to code for the first time. They are just like us, know nothing about programming at the beginning. Our computers are the same as their computers. I believe that if they can, we can. Computing business is depending hugely on your idea. If you have a great idea, but you can not make it real, you can’t translate what’s in your head to a program, then it will only be in your head. These people, they have some good ideas, like Facebook, Google or Dropbox, and they just made it happen by coding. Many of the richest men on earth started their fortune with simple programming. Google co-founders start Google program in their dormitory room. Facebook has its beginning in campus. You don’t have to have big computer, or big budget to start building the coolest companies like Google or Facebook.

Programming is actually very simple. Each programming language contains a set of simple commands. Each command is simple but by putting these simple commands together, you can solve extremely complex problems. It is like playing chess. Each chess piece has very simple movement pattern. You can learn to play chess very quickly in half an hour. But to be a good chess player, you probably need years of experience. It’s more of an art than a science. Programming is the same.

OK, now you know that programming is great! It’s cool! Where do we start? We start with this chapter, of course.
What is Python?

We can code using many different computer languages. We do have many of them today, languages such as C, C++, Java, PHP, Basic or even Fortran are all in use.

But for this book, we have chosen Python as our main programming language. Python is a very modern computer language, it was developed recently so the difficulties that we found in the older languages like C are reduced or eliminated.

Python (http://www.python.org/about/gettingstarted/) is introduced and it is now widely accepted by many universities (such as MIT, http://tinyurl.com/dxk7bsg) as an entry point for simple programming concept (as reference from the graph to the left).

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

**Python is powerful and fast** Fans of Python use the phrase "batteries included" to describe the standard library, which covers everything from asynchronous processing to zip files. The language itself is a flexible powerhouse that can handle practically any problem domain. Build your own web server in three lines of code. Build flexible data-driven code using Python's powerful and dynamic introspection capabilities and advanced language features such as meta-classes, duck typing and decorators.

Python lets you write the code you need, quickly. And, thanks to a highly optimized byte compiler and support libraries, Python code runs more than fast enough for most applications.

**Python runs everywhere** Python is available for all major operating systems: Windows, Linux/Unix, OS/2, Mac, Amiga, among others. There are even versions that run on .NET and...
the Java virtual machine. You’ll be pleased to know that the same source code will run unchanged across all implementations.

**Python is friendly... and easy to learn** The Python newsgroup is known as one of the friendliest around. Python also comes with complete documentation, both integrated into the language and as separate web pages. Online tutorials target both the seasoned programmer and the newcomer. All are designed to make you productive quickly. The availability of first-rate books completes the learning package.

**Python is Open** The Python implementation is under an open source license that makes it freely usable and distributable, even for commercial use. The Python license is administered by the Python Software Foundation. Most of the modern platform such as Google App Engine supports Python.

There are lots of online materials available for Python learners. The ones I particularly want to mention is http://www.learnpython.org/ and http://www.tutorialspoint.com/. With simple explanations and realistic examples. These 2 webs are highly recommended. Also a live coding window where you can put the python code and see the live results can be found there.

**Download Python**

So, we will talk to computer using a Python language, but what does it really mean? It means you needs Python interpreter on your computer, so we can write our commands onto this interpreter’s editor and it will translate to computer tongue. There many editors for Python, most of them are free but the one that we will use is IDLE which is the default Python editor. Normally it already comes with every Mac or Linux computers. But if you are using Windows or the version on your Mac is not up to date then you want to download and install it on your computer first. Head for this website, http://www.python.org/download/ and choose the program that is appropriate to your platform. We will use version 3.4.1 as our default.

After you finish your install, you should be able to call idle or double click on it. Don’t worry about a book or manual, if you can connect to the internet, everything you need about the programme support and documentation can be found on the Python website, www.python.org.
Hello World!

OK, everything is set. Let’s do something with it.

The first program that all the experience coders coded is Hello World program. We just want to order the computer to display this simple sentence to the screen. And the command that I want you to know is the print command. It will just print the message you want to display onto the screen. OK, let’s have a look at some examples.

```python
>>> print("Hello world!")
Hello world!

>>> print("Superman")
Superman
```

This command is simple. It will just display anything in between double quotes “ ” to the screen. (Note that you can use single quotes ’ ’ in place of double quotes “”, in case you need to display ”)

Commands could be sent to computer through Python in 2 different ways. The first is the way that I just demonstrated, sending a command in the main window which is called the command window. This way you type your one command at a time, just right after the command prompt, >>>>, and press enter. The Python interpreter will translate it and respond with appropriate action. This is good because you can use it for quick calculations like this one,

```python
>>> 25 + 30
55
```

But this will be very slow for a big program where you need to send many or hundreds of commands to the editor. If that’s the case, you can actually put all your commands in a file. With the list of commands in the file, you can edit it, save it and send it to run. You can create a new file with the editor’s File > New menu. You get a blank editor window, then, all the commands can be added here, like this following example.

```
print("Peter Pan")
print("Super Man")
print("Spider Man")
```
You can save the file and HOORAY! that’s your first program! You already, officially, are a programmer. The type for this file is `.py` and that’s for Python. OK, now you can run the program. You can either choose Run from the menu or press F5 on your keyboard. And the result of your program will be shown in the command window.

Let’s do something more fancy. Let’s print a Christmas tree to the screen with this code.

```
print ("    *")
print ("   **")
print ("  ***")
print (" ****")
print ("*****")
print ("     ")
```

Now you can do lots of things here. Let’s create your program and play with it. If you want to explore further, try challenge yourself with some problems in exercise section.

With your playing, you probably notice that the command `print("")` is very strict. It will give you various error messages if you misspelled the command or even forgot a tiny `. I told you the you computer is not very clever. It does exactly what it’s told. Even when you order it with this command,

```python
>>>print ("23+34")
23+34
```

It will just print exactly everything inside" ". If you intended to get the result of calculation, you will have to use the print command without " ". To do this, everything between ( ) will be considered as an unfinished statement and need to be processed first before printing the result to the screen.

```python
>>>print (23+34)
57
```

Be careful if the statement between ( ) can’t be processed, like this following example, then you will get an error message like this one.

```python
>>>print (HelloWorld!)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print (HelloWorld)
NameError: name 'HelloWorld' is not defined
```
OK, one last thing for this first and easiest introduction to the **print** command is that you have to print out the result of calculation otherwise everything is done in the background and they will never show up for the user. Try save this example in a new .py file and run it.

```python
print("Hello World!")
25+30
print("Hello World!")
Hello World!
Hello World!
```

You will only see Hello World! twice but no 55. The command 25+30 did happen but in the background and if you didn’t order the computer to print the result on the screen it will just remain in the background (which will be extremely convenient when you have a long calculation and don’t want to display every single step of that calculation.)

**Extra material**

Normally the command `print(`"`)` will end the line with a special character “\n” which means a new line. That’s why when you call the print commands for 3 times, it will print result in 3 separate lines as in the following example on the left. We can change this behaviour by providing the 2nd parameter to the print command, **end**. This parameter is to specify how the print command will be ended. In the example on the right, we give the 3rd and 4th command with “,” ending and the 5th command with a full stop ending as shown in the result.

```python
print("Hello World!")
print("Hello World!")
print("Hello World!")
print("Superman")
print("Superman")
print("Superman", end="", ")
print("Superman", end="", ")
print("Superman", end="." )
Superman
Superman
Superman, Superman, Superman.
```
Conclusion

That’s it for the introduction and your very first program. In this chapter, you learned the easiest basic of Python programming with one commands, print(“ “). You can order computer to do things with specific commands in strict patterns. After this one, you will start to know more of the Python commands and put them together for more complex tasks. I hope that after this class you get some inspiration and want to start writing program. In the next chapter, we will have even more fun with some drawing and painting, of course, on computer with Python.
Exercises

1. Write a program to print names of 4 persons sitting next to you.

2. Write a program to display a car made only of ‘*’.

3. Write a program to show the ID, name and surnames of 5 people sitting next to you in the following format.

   My name is : Theo Walcott
   ------------------------
   1: 560000001 Peter Pan
   2: 560000002 Harry Potter
   3: 560000021 Pat The Cat
   4: 560000201 Stan The Great
   5: 560000031 Super Man

4. Output this shape on the screen:

   **********
   *********
   ********
   *******
   ******
   *****
   ****
   ***
   **
   *

5. Write a program that prints name of your favourite movie characters in double quotes. The program output should look like this:

   My most favourite is “Superman”.
   I also like “Doraemon”
   And also “Harry Potter”

6. Write a program to display this little bunny:

   (\ /)
   (. .)
   O(") (")

7. Write a program to show result of this calculation:

   \[4.3533 + 233.23 - 823.144 \times 5 + 21.212\]
Chapter 2

Python and Turtles

Python Turtle

OK, you can now write some trivial programs. Now let do something a bit more fun. There is another great feature of the Python program I want to show you. It's the extensible nature of the Python. We can extend the core ability of Python by adding libraries into the program. We use IMPORT command when we want to add some libraries of commands into the program. For example we can add ability to do some complex mathematical calculations with import math and after that point you can use all the functions in the added library. We will discuss about Math functions pretty soon but let's begin now with something fun, the Turtle library.

So let's begin your program with this line,

import turtle

After this command is called, you will have all functionality of a turtle.

What is Python's Turtle? (The name sounds pretty funny now). Turtle is a drawing tool that can be controlled by specific turtle commands. The commands are used to controlled the
movement of the turtle (or pen’s tip), or control the setting of colour or size of the pen. Some examples of the commands are shown here,

**Turtle’s movement**

`turtle.forward(distance)` or `turtle.fd(distance)`  
Move the turtle forward by the specified distance, in the direction the turtle is headed.

`turtle.back(distance)` or `turtle.bk(distance)` or `turtle.backward(distance)`  
Move the turtle backward by distance, opposite to the direction the turtle is headed. Do not change the turtle’s heading.

`turtle.right(angle)` or `turtle.rt(angle)`  
Turn turtle right by angle units. (Units are by default degrees, but can be set via the `degrees()` and `radians()` functions.) Angle orientation depends on the turtle mode, see `mode()`.

`turtle.left(angle)` or `turtle.lt(angle)`  
Turn turtle left by angle units. (Units are by default degrees, but can be set via the `degrees()` and `radians()` functions.) Angle orientation depends on the turtle mode, see `mode()`.

`turtle.goto(x, y)` or `turtle.setpos(x, y)` or `turtle.setposition(x, y)`  
Move turtle to an absolute position. If the pen is down, draw line. Do not change the turtle’s orientation.

`turtle.setx(x)`  
Set the turtle’s first coordinate to x, leave second coordinate unchanged.

`turtle.sety(y)`  
Set the turtle’s second coordinate to y, leave first coordinate unchanged.

`turtle.setheading(to_angle)` or `turtle.seth(to_angle)`  
Set the orientation of the turtle to to_angle. Here are some common directions in degrees: 0 - east, 90 - north, 180 - west, 270 - south

`turtle.home()`  
Move turtle to the origin – coordinates (0,0) – and set its heading to its start-orientation

`turtle.circle(radius, extent=None, steps=None)`  
radius – a number, extent – a number (or None), steps – an integer (or None)  
Draw a circle with given radius. The center is radius units left of the turtle; extent – an angle – determines which part of the circle is drawn. If extent is not given, draw the entire circle. If
extent is not a full circle, one endpoint of the arc is the current pen position. Draw the arc in
counterclockwise direction if radius is positive, otherwise in clockwise direction. Finally the
direction of the turtle is changed by the amount of extent. As the circle is approximated by an
inscribed regular polygon, steps determines the number of steps to use. If not given, it will be
calculated automatically. May be used to draw regular polygons.

**turtle.dot(size, color)**
size – an integer >= 1 (if given), color – a colorstring or a numeric color tuple
Draw a circular dot with diameter size, using color. If size is not given, the maximum of
pensize+4 and 2*pensize is used.

**turtle.speed(speed=None)**
Parameters: speed – an integer in the range 0..10 or a speedstring (see below)
Set the turtle's speed to an integer value in the range 0..10. If no argument is given, return
current speed.
"fastest": 0, "fast": 10, "normal": 6, "slow": 3, "slowest": 1

**Pen control**

**turtle.pendown() or turtle.pd() or turtle.down()**
Pull the pen down – drawing when moving.

**turtle.penup() or turtle.pu() or turtle.up()**
Pull the pen up – no drawing when moving.

**turtle.pensize(width) or turtle.width(width)**
Set the line thickness to width or return it. If resizemode is set to “auto” and turtleshape is a
polygon, that polygon is drawn with the same line thickness. If no argument is given, the
current pensize is returned.

**turtle.pencolor(args)**
Return or set the pencolor.
Set pencolor to colorstring, which is a Tk color specification string, such as "red", "yellow", or
"#33cc8c".

pencolor((r, g, b))
Set pencolor to the RGB color represented by the tuple of r, g, and b. Each of r, g, and b
must be in the range 0..colormode, where colormode is either 1.0 or 255 (see colormode()).
pencolor(r, g, b)
Set pencolor to the RGB color represented by r, g, and b. Each of r, g, and b must be in the range 0..colormode.

fillcolor(colorstring)
Set fillcolor to colorstring, which is a Tk color specification string, such as "red", "yellow", or "#33cc8c".

turtle.fill(flag)
Call fill(True) before drawing the shape you want to fill, and fill(False) when done. When used without argument: return fillstate (True if filling, False else).

turtle.begin_fill()
Call just before drawing a shape to be filled. Equivalent to fill(True).

turtle.end_fill()
Fill the shape drawn after the last call to begin_fill(). Equivalent to fill(False).

turtle.reset()
Delete the turtle’s drawings from the screen, re-center the turtle and set variables to the default values.

turtle.clear()
Delete the turtle’s drawings from the screen. Do not move turtle. State and position of the turtle as well as drawings of other turtles are not affected.

turtle.write(arg, move=False, align="left", font=("Arial", 8, "normal"))
arg – object to be written to the TurtleScreen
move – True/False
align – one of the strings "left", "center" or right"
font – a triple (fontname, fontsize, fonttype)
Write text - the string representation of arg - at the current turtle position according to align ("left", "center" or right") and with the given font. If move is true, the pen is moved to the bottom-right corner of the text. By default, move is False.

turtle.hideturtle() or turtle.ht()
Make the turtle invisible. It’s a good idea to do this while you’re in the middle of doing some complex drawing, because hiding the turtle speeds up the drawing observably.

turtle.showturtle() or turtle.st()
Make the turtle visible.
Tell Turtle’s state

turtle.position() or turtle.pos()
Return the turtle’s current location (x,y)

turtle.towards(x, y)
Return the angle between the line from turtle position to position specified by (x,y), the vector or the other turtle. This depends on the turtle’s start orientation which depends on the mode - “standard”/”world” or “logo”).

turtle.xcor()
Return the turtle’s x coordinate.

turtle.ycor()
Return the turtle’s y coordinate.

turtle.heading()
Return the turtle’s current heading (value depends on the turtle mode, see mode()).

turtle.distance(x, y=None)
Return the distance from the turtle to (x,y), the given vector, or the given other turtle, in turtle step units.

Settings for measurement

turtle.degrees(fullcircle=360.0)
Set angle measurement units, i.e. set number of “degrees” for a full circle. Default value is 360 degrees.

turtle.radians()
Set the angle measurement units to radians. Equivalent to degrees(2*math.pi).
Explore the full functionality of the turtle on the Python official site at https://docs.python.org/2/library/turtle.html, some useful functions are listed here.

<table>
<thead>
<tr>
<th>Turtle motion</th>
<th>Pen control</th>
</tr>
</thead>
<tbody>
<tr>
<td>forward()</td>
<td>fd()</td>
</tr>
<tr>
<td>backward()</td>
<td>bk()</td>
</tr>
<tr>
<td>right()</td>
<td>rt()</td>
</tr>
<tr>
<td>left()</td>
<td>lt()</td>
</tr>
<tr>
<td>goto()</td>
<td>setpos()</td>
</tr>
<tr>
<td>setx()</td>
<td></td>
</tr>
<tr>
<td>sety()</td>
<td></td>
</tr>
<tr>
<td>circle()</td>
<td></td>
</tr>
<tr>
<td>dot()</td>
<td></td>
</tr>
<tr>
<td>degrees()</td>
<td></td>
</tr>
</tbody>
</table>

The turtle is initialised on the drawing board at location (0,0) and heading in 0 degree direction. The coordinates are normal cartesian coordinates. You can move the turtle and see the line on its trail. With this simple setting, you can use it to draw whatever you can imagine.

Now let's start drawing something. How about a rectangle? To draw a rectangle is pretty simple, just draw a straight line, then turn 90 degree and draw another straight line. Keep on doing this for 4 iterations and we will get a rectangle.

```python
import turtle
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
```

Save the program and run it, you will see the turtle drawing a rectangle slowly on your screen. If you get tired of typing `turtle`, then you can give your turtle a nickname. Any name you would like to call it. For example `import turtle as t` or even `import turtle as kitty`. Then use that name instead of the word `turtle`.

Now let do something a little bit more complex. In this example, we will see the usage of commands to change the size of the pen tip and also the colour. In the program, we import the turtle with a shorter name, t. We then set the size of the pen to 10 pixels. We will draw a circle with a radius of 100 pixels and then turn left for 90 degrees so that we can draw another circle with a different orientation. We change the pen colour every time before we draw a new circle. We can set colour to a specific colour names like red, green, blue or we can use HTML colour code (more of the colour can be found here, http://

Python for Beginners
www.w3schools.com/html/html_colors.asp). And Keep doing this for 4 times and we get the output displayed below.

```
import turtle as t
    t.pensize(10)
    t.pencolor("red")
    t.circle(100)
    t.left(90)
    t.pencolor("blue")
    t.circle(100)
    t.left(90)
    t.pencolor("yellow")
    t.circle(100)
    t.left(90)
    t.pencolor("#FF55FF")
    t.circle(100)
```

How about something even fancier, like a Mickey mouse. With this one, we have to do a little bit of planning. A Mickey mouse will consist of 3 filled circle or 3 dots. One dot in the centre and two smaller dots on the upper left and upper right. Remember, when the turtle is initialised, it is placed at coordinates (0,0) on a cartesian plane. At that point we place a dot of size 200 pixels. We then move our turtle to the coordinates (-90,90) which is 90 pixels to the left and and 90 pixels up compared to the centre (0,0)

```
import turtle as kitty
    kitty.dot(200,"black")
    kitty.setpos(-90,90)
    kitty.dot(100,"black")
    kitty.setpos(90,90)
    kitty.dot(100,"black")
```

Let’s try another example by applying colours to many half-circles to make colourful shape. Normally when we call a command `turtle.circle(100)`, the turtle will draw a full circle. But if we just want to draw part of a circle, then we have to specify the second parameter, to indicate the angle of the circle we need to draw. (If that number is 360, it will draw a full circle, of course). This time we are going to use `turtle.circle(100,180)` command to draw an arc of half a circle. To get us a complete enclosed half circle (not just an arc), we instruct the turtle to draw a straight line back to the original (0,0). Then we fill that half circle with what ever colour we want. I have chosen, blue, yellow, #FE642E and #FF0040. The last two colours
are HTML colour code, which I picked from http://html-color-codes.info/. Then keep doing the same thing for 4 times with different turtle’s headings (so that we can have the half circle in different orientations) and you will get the shape of the right of the following code.

```
import turtle as t

t.setheading(0)
t.fillcolor("blue")
t.begin_fill()
t.circle(50,180)
t.goto(0,0)
t.end_fill()

t.setheading(90)
t.fillcolor("yellow")
t.begin_fill()
t.circle(50,180)
t.goto(0,0)
t.end_fill()

t.setheading(180)
t.fillcolor("#FE642E")
t.begin_fill()
t.circle(50,180)
t.goto(0,0)
t.end_fill()

t.setheading(270)
t.fillcolor("#FF0040")
t.begin_fill()
t.circle(50,180)
t.goto(0,0)
t.end_fill()
```

OK, one last example for the turtle. Now I want to draw a BMW logo. We begin with the setting of the size of the pen to 4 for the line thickness and set the line colour to #C0C0C0 (that’s colour of silver. I found it by Google “Silver HTML colour code”). The BMW logo has a black circle background, so we draw a big black dot with size of 300 pixels (so it has a radius of 150 pixels). We then have to conjure another 4 quarters of circle with blue and white alternately. Let’s draw the first one by moving the turtle for 100 pixels to the right (original heading of the turtle is set to 0, meaning “right”). We then draw a 90-degree arc of radius 100 pixels with command `circle(100, 90)`. We then draw a straight line back to the original (0,0) to close the arc and fill it with blue. For the other 3 quarters we just do the same thing with different turtle headings and different filling colours.
Now I think you can see lots of possibilities popping up in your imagination. Let’s draw those fancy shapes in your head with Python’s turtle. If you still have no idea what to try then head straight to the exercise section, there will be many shapes for you to start drawing.

```python
import turtle as t

t.pencolor("#C0C0C0")
t.pensize(4)
t.dot(300,"Black")

t.fillcolor("white")
t.begin_fill()
t.fd(100)
t.setheading(90)
t.circle(100,90)
t.goto(0,0)
t.end_fill()

t.setheading(90)
t.fillcolor("blue")
t.begin_fill()
t.fd(100)
t.setheading(180)
t.circle(100,90)
t.goto(0,0)
t.end_fill()

t.setheading(180)
t.fillcolor("white")
t.begin_fill()
t.fd(100)
t.setheading(270)
t.circle(100,90)
t.goto(0,0)
t.end_fill()

t.setheading(270)
t.fillcolor("blue")
t.begin_fill()
t.fd(100)
t.setheading(360)
t.circle(100,90)
t.goto(0,0)
t.end_fill()
```
Exercises

1. Write a program with Turtle to display this shape (red box in circle) on the screen:

![Red box in circle](image)

2. Write a program with Turtle to display this shape (man) on the screen:

![Man](image)

3. Write a program with Turtle to display this shape (star) on the screen:

![Star](image)

4. Write a program with Turtle to display this shape (face) on the screen:

![Face](image)
5. Write a program with Turtle to display this shape (rings) on the screen:

![Rings Diagram]

6. Write a program with Turtle to display this shape (Chain) on the screen:

![Chain Diagram]

7. Write a program with Turtle to display this shape (House) on the screen:

![House Diagram]

8. Use turtle to draw logos of Mitsubishi and Toyota
Chapter 3
Variables

One of the most useful and powerful concept of programming is **variable** and that’s what we are going to mainly focus in this chapter.

In the previous chapters, you ordered the computer to do something with specific commands and specific values. For example, you created a square with size of 100 pixels. You drew circles with specific radiuses. You put dots on specific locations. Imagine that we are going to write program that let our users specify what size of the square she wants. So at the time you write your program, you don’t know exactly what size of the square the user wants. During programming, you can’t say `turtle.circle(100)` in your program anymore because 100 might not be what the user wants. We have to say something like, `turtle.circle(x)` instead, where `x` is the unknown value the user is going to give us when the program is run on her computer (which of course might not be the same computer you use to write that program). So, `x` is what we call a **variable**.
Basically, a variable is a place in computer’s memory where you keep value. It will come into life when you declare it. By declaration, the computer will reserve memory locations to store values. This means that when you create a variable you reserve some space in memory.

Depends on what type of data you want for that variable, the Python interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

**Declaring and Assigning Values to Variables:**

Python variables do not have to be explicitly declared to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. You have 2 operands on both sides of the equal sign. The one on the left of the = operator is the name of the variable and the one on the right of the = operator is the value to be stored in that variable.

```
x = 100
peter  = 2.4345
headmaster = "John"
```

`x`, `peter` and `headmaster` are the names of variables. You assigned 100 (which is an integer), 2.4345 (which is a floating point) and John (which is a string) to those variables respectively.

**Naming**

Variable name can be nearly anything you want, of course, with a few exceptions. Those common exceptions are

- you cannot begin variable name with number
- variable name cannot contain space
- Most of special characters are prohibited (Underscore _ is OK)

And one important thing to remember about variable is that it’s case sensitive. Variable with the different cases are considered different variables event though they are the same name.
For example, peter, Peter and PeTer are 3 different variables. You have 5, 12 and 10 as results in the following example:

```
Peter = 5
PeTer = 10
peter = 12
print(Peter)
print(peter)
print(PeTer)
```

To access the value in the variable, you just refer to the variable by its name. For example, if you want to print the value that x stores, you say, print(x) and you will see 100 on the screen. Be careful, print(“x”) will give different output. This latter will print a character x on the screen instead of 100 because it will print exactly everything inside “”, remember?

Saying print(peter) and print(headmaster) will have similar effect. You will see 2.4345 and John on the screen.

**Changing Values:**

Each variable can hold just one value, so if the same variable is reassigned with the new value, it will just replace the old one. In the following example, variable x is initialised with value 1 and later it is changed to 15 before it is changed again to 10. It's then printed. So we will see 10 in our output at this point. x is later changed to 5 before it is printed again. So we will have 5 too.

```
x = 1
x = 15
x = 10
print (x)
x = 5
print (x)
```

As previously mentioned, there are many types of variable. For example, a person’s age is stored as a numeric value and his name and address is stored as alphanumeric characters. Python has various standard types that are used to define the operations possible on them and the storage method for each of them. A few of them are just introduced here.
• Integer variable - a variable that keeps whole number
• Floating point variable - a variable that keeps number with decimal point
• String variable - a variable that keeps series of characters (or text)
• Boolean variable - a variable that keeps True and False

Python is clever enough to change the type of the variable automatically. You can assign a floating point to a variable that previously keeps integer value and Python will change the type of that variable to floating point automatically. This is also true for other types of variables. Look at this following example

```
x = 1
print (x)
x = 15.2
x = “Peter pan”
print (x)
```

## Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example:

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example:

```
a, b, c = 1, 2, “John”
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b, and one string object with the value "John" is assigned to the variable c.

## Mathematical Operations

You can use variables in the same way that you normally used the numbers in calculations. For example:
There are many mathematical operators that can be used in simple calculation like addition, subtraction, multiplication and so on. The table here shows the programming symbols for that calculations.

Assume that a = 10 and b = 20, these are examples of the operations and results.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition - Adds values on either side of the operator</td>
<td>a + b will give 30</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction - Subtracts right hand operand from left hand operand</td>
<td>a - b will give -10</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication - Multiplies values on either side of the operator</td>
<td>a * b will give 200</td>
</tr>
<tr>
<td>/</td>
<td>Division - Divides left hand operand by right hand operand</td>
<td>b / a will give 2</td>
</tr>
<tr>
<td>%</td>
<td>Modulus - Divides left hand operand by right hand operand and returns remainder</td>
<td>b % a will give 0</td>
</tr>
<tr>
<td>**</td>
<td>Exponent - Performs exponential (power) calculation on operators</td>
<td>a**b will give 10 to the power 20</td>
</tr>
<tr>
<td>//</td>
<td>Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.</td>
<td>9/2 is equal to 4 and 9.0/2.0 is equal to 4.0</td>
</tr>
</tbody>
</table>

Error from data type difference

However, mixing different types of variables in one calculation will cause a variety of responses ranging from unexpected results to errors.

Try adding a number to a string and you will see the effect. In the following example, variable `x` is an integer when a variable `y` is a string (notice the double quote). `x + y` will then spit an error message. Just don’t panic, read the error message and that can be easily identified.
Type cast

Python provides a clever way to change the type of variables by type-casting them with special commands, int(), float(), str(), and bool(). The names of the commands clearly explain themselves. See the following example.

```
>>> x = 1.532
>>> int(x)
1
>>> float(x)
1.532
>>> str(x)
'1.532'
>>> bool(x)
True
```

With the previous example, \texttt{x} is a floating point variable containing a value 1.532. Note that when \texttt{int(x)} is called, the output from that function call is an integer, however the original \texttt{x} still remains a floating point. But if you say something like, \texttt{y = int(x)}, this case, \texttt{x} is still a floating point, but \texttt{y} is an integer.

Input

Usually, we use variables at the place in the program where we do not know the value at the time of programming. For example, we need user’s input for calculation. That’s where we need another command to get input from user, in this case, from a keyboard.

```
x = input("Please input a number: ")
```

The command for input is again very simple. Keyword \texttt{input} follows by ("xxxx") with
whatever text you want to display while waiting for user’s response. With the following example, it just displays the text in double quotes and waits. After we input something, a number or text, we then hit enter and that value will be kept in the variable \( x \). In this particular case \( x \) is “20”.

```
>>> x = input("How old is Superman? :")
How old is Superman? :20
```

**Important note!** Variable that directly takes value from input is a **string** variable. So it can’t be used directly in mathematical operations. This following example is very common mistake.

```python
x = input("How many dollars do you want to buy? ")
y = input("Exchange rate (Baht per dollar) :")
payment = x * y
print("you have to pay ",payment," Baht")
```

```
Traceback (most recent call last):
  File "/Users/apple/Documents/xx.py", line 3, in <module>
    payment = x * y
TypeError: can’t multiply sequence by non-int of type 'str'
```

The problem can be solves easily with the typecast functions. The correct program will look like this:

```python
x = int(input("How many dollars do you want to buy? "))
y = float(input("Exchange rate (Baht per dollar) :"))
payment = x * y
print("you have to pay ",payment," Baht")
```

```
How many dollars do you want to buy? 20
Exchange rate (Baht per dollar) : 32.2
you have to pay 644 Baht
```

We enclosed the input commands in within typecast functions, \( int() \) and \( float() \) to change \( x \) and \( y \) to integer and a floating point variable respectively.

**Multiple inputs**

OK, you have come this far in this chapter, so here is your bonus. After you use the \( \text{input()} \) command for sometimes you can notice that you can get only 1 variable per 1 input command. And that can be annoying if we have to take many inputs in the program. With that you have to use as many input commands. “Can I get many inputs with one input
command?”, you may ask. Of course, you can. With a few click on the web you will see `input().split()`. That sounds like an alien language now. We will talk about that later but, for now, this bonus command is to split, of course, the input into pieces and assign to multiple variables with that one single command. Look at this following example:

```python
>>> x, y = input("gimme two numbers: ").split()
gimme two numbers: 10 12
>>> x
'10'
>>> y
'12'
```

You can use it to split many values at once, but be very careful, the number of variables to the left of = must be equal to the number of input provided by user, otherwise you will get the error like this following example:

```python
>>> a, b, c = input("Three numbers please: ").split()
Three numbers please: 2 12 35 24
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    a, b, c = input("Three numbers please: ").split()
ValueError: too many values to unpack (expected 3)
```

## Formatted outputs

This is your second bonus. You can actually print out the values within the variables along with normal text by separate them with , or + in the print command. In the following example, we print a sentence “My name is” and follows by value in x which is “Peter” and then print a space follows by a value in variable y, which is “Pan”

```python
>>> x = "Peter"
>>> y = "Pan"
>>> print("My name is ", x," ", y)
My name is Peter Pan
```
You can also format how a floating point number can be displayed using a \%.f placeholder. A number in front of f indicating number of decimal points to be printed.

<table>
<thead>
<tr>
<th>commands</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>\print(&quot;%.2f&quot; % 30.00123)</td>
<td>30</td>
</tr>
<tr>
<td>\print (&quot;%.3f&quot; % 30.00163)</td>
<td>30.002</td>
</tr>
<tr>
<td>x = &quot;peter pan&quot;</td>
<td></td>
</tr>
<tr>
<td>print(&quot;My name is &quot;+ x)</td>
<td>My name is peter pan</td>
</tr>
</tbody>
</table>
Exercises

1. Write a program to get 3 inputs from user and print them in reversed order:

   Enter the first number: 1
   Enter the second number: 2
   Enter the third number: 3

   Reverse order of 1 2 3 --> 3 2 1

2. Write a program that takes two integer numbers and prints out the product (multiplication) of the first and the second numbers. The program output should look like this:

   Enter the first integer number: 20
   Enter the second integer number: 5

   20*5 = 100

3. Write a program that converts a volume in milliliter to fluid ounce (1 milliliter = 0.034 ounces). The program output should look like this:

   Enter a volume (in ml): 250

   Volume in milliliter is 250.00
   Volume in ounce is 8.50

4. Write a program that converts a temperature in Celsius (C) to Fahrenheit (F). The program output should look like this:

   Enter a temperature in C: 37

   Temperature in C = 37.00
   Temperature in F = 98.60

5. Write a program which receives three numbers a, b, c from user, and prints out the area of triangle with edges a, b, c.

   Enter length of 3 sides of a triangle
   Side A: 3
   Side B: 4
   Side C: 4

   The area is: 6.0
6. Write a program to take two integer numbers as input data and display their sum, difference, product (multiplication). The program output should look like this:

```
Enter the first integer number: 8
Enter the second integer number: 22

The sum is 30
The difference is -14
The product is 176
```

7. Write a program that receives a three-digit integer from the user, then print out the sum of all these three digits. Your program should respond as follows:

```
x=123
Sum of digit = 6
```

8. Write a program to ask for a radius and colour of a circle and draw the circle at the middle of the screen.

```
Enter the radius of circle: 100
Enter the color (red, blue, green): blue
```

9. Asks the positions (x,y) and radius of two circles from user. Draws these two circles and draws a dot at the middle point between them. (Use Turtle Module)

```
Enter radius: 50
Enter the centre of 1st circle: -50 0
Enter the centre of 1st circle: 100 150
```
Chapter 4

Predefined functions

You can already do lots of things with just three previous chapters. Lots of calculations. Lots
of drawings. You also may have noticed that, with help from some special functions
like .split() or int( ), you can do even a lot more. Those functions are called predefined
functions, the functions that has been included or built by someone and can be later
included into our program. There are myriad of those predefined functions out there. Too
many of them so that I am pretty sure that you cannot use or even cannot remember them
all. Turtle functions that you have seen and used in the first chapter are also predefined
functions. In this chapter, I’ll introduce a set of mathematical functions. The rest will be up to
you to explore and use.

Maths functions are in Python’s standard library (https://docs.python.org/3/library/
index.html). This library is Python’s native built-in functions which actually means they are
always available and you don’t have to download or config anything else for this commands.
You can just use them.

To start with, let’s have a look at all mathematical functions listed herehttps://
docs.python.org/3/library/math.html.
**Math**

Similar to Turtle, to add mathematical functions to your program we need to start by

```python
import math
```

With this one line of code, you add mathematical functionalities such as Power and logarithmic functions or Trigonometric functions to your program.

**Number-theoretic and representation functions**

- `math.ceil(x)` returns the ceiling of x, the smallest integer greater than or equal to x.
- `math.fabs(x)` returns the absolute value of x.
- `math.factorial(x)` returns x factorial. Raises ValueError if x is not integral or is negative.
- `math.floor(x)` returns the floor of x, the largest integer less than or equal to x.
- `math.trunc(x)` returns the Real value x truncated to an Integral (usually an integer).

**Power and logarithmic functions**

- `math.exp(x)` return e**x.
- `math.log(x[, base])`
  
  With one argument, return the natural logarithm of x (to base e).
  
  With two arguments, return the logarithm of x to the given base, calculated as log(x) / log(base).
- `math.log2(x)` returns the base-2 logarithm of x. This is usually more accurate than log(x, 2).
- `math.log10(x)` returns the base-10 logarithm of x. Usually more accurate than log(x, 10).
- `math.pow(x, y)` returns x raised to the power y.
- `math.sqrt(x)` returns the square root of x.
### Trigonometric functions

- `math.acos(x)` returns the arc cosine of `x`, in radians.
- `math.asin(x)` returns the arc sine of `x`, in radians.
- `math.atan(x)` returns the arc tangent of `x`, in radians.
- `math.cos(x)` returns the cosine of `x` radians.
- `math.sin(x)` returns the sine of `x` radians.
- `math.tan(x)` returns the tangent of `x` radians.

### Angular conversion

- `math.degrees(x)` converts angle `x` from radians to degrees.
- `math.radians(x)` converts angle `x` from degrees to radians.

### Constants

- `math.pi` The mathematical constant $\pi = 3.141592\ldots$, to available precision.
- `math.e` The mathematical constant $e = 2.718281\ldots$, to available precision.
Exercises

1. Many foreigners have difficulties in understanding the traditional Thai system to measure the area of a piece of land. In the past, Thai people used terms such as Rai, Ngan, and Tarangwa. Your task is to create a program that takes the inputs of the area of a piece of land described using these three traditional Thai units to output the total area of this piece of land in square meters. The conversion table and examples of the program interface is as follow.

<table>
<thead>
<tr>
<th>Conversion Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Rai = 4 Ngan = 400 Tarang Wa = 1600 square meters</td>
</tr>
<tr>
<td>1 Ngan = 100 Tarang Wa = 400 square meters</td>
</tr>
<tr>
<td>1 Tarang Wa = 4 squared meters</td>
</tr>
</tbody>
</table>

Input the number of Rai: 2
Input the number of Ngan: 0
Input the number of Tarangwa: 40

The total area of the is is 3360 square meters

2. Write a program to find the distance, d, between points A and B where the positions of A and B are specified by the Cartesian coordination of (x1, y1) and (x2, y2), respectively.

\[ d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \]

Enter the first point: 1 1
Enter the second point: 5 5

The distance between (1, 1) and (5, 5) is 5.66.

3. The period of one swing of a simple pendulum, T, is given by

\[ T = 2\pi \sqrt{\frac{l}{g}} \]

where (in metric units) \( T \) = period (sec), \( l \) = length of pendulum (m), \( g \) = gravitational acceleration = 9.81 m/sec\(^2\)

Write a program capable of computing the period, T, for user inputting the length of a
pendulum. The program output should look like this:

| Enter the length of a pendulum in meter: 0.5 |
| Period (T): 1.4 |

4. Write a program that gets 2 inputs from user, a deposit amount and interest rate in percent, and then computes yearly interests earned for three years. Assume that the earned interests are summed into the deposit amount every year. The program output should look like this:

| Input amount: 100 |
| Yearly interest rate: 5 |
| At the end of year 1 - Interest: 5.0 new balance: 105.0 |
| At the end of year 2 - Interest: 5.25 new balance: 110.25 |
| At the end of year 3 - Interest: 5.5125 new balance: 115.7625 |

5. Write a program that takes the three coefficients of a quadratic equation, a, b and c and find its roots.

\[ root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \]

Note that: the program may give error if this term, \( b^2 - 4ac \) is less than 0, because the root is imaginary and cannot be computed. Just don't worry about it for now. The program output should look like this:

| Enter a value for a: 1 |
| Enter a value for b: -3 |
| Enter a value for c: -4 |

The roots are 4 and -1.
6. Write a program that computes duration of projectile's flight and its height above the ground when it reaches the target.

**Program Constant**

\[ g = 32.17 \text{ ft/s}^2 \quad /* \text{gravitational constant} */ \]

**Problem Inputs**

- `double theta` /* input angle of elevation (in radians) */
- `double distance` /* distance (in feet) to target */
- `double velocity` /* projectile velocity (ft/s) */

**Problem Output**

- `double time` /* time of a flight (in seconds) */
- `double height` /* height at impact */

**Relevant Formulas**

\[
\text{time} = \frac{\text{distance}}{\text{velocity} \times \cos(\theta)}
\]

\[
\text{height} = \text{velocity} \times \sin(\theta) \times \text{time} - \frac{g \times \text{time} \times \text{time}}{2}
\]

```
Input an angle in radian: 0.5236
Input a distance to target: 100
Input a projectile velocity: 80

The flight will take 1.44 seconds.
```

7. Write a program that calculates the area of the trapezoid.

```
\text{area} = \frac{a+b}{2} \times h
```

```
Enter a: 3
Enter b: 5
Enter h: 2.5

The area of trapezoid is 10.00
```
8. You are designing a new aquarium that contains a walkway for visitors to walk through the aquarium to see different kinds of animals in the aquarium, and you want to know how much water is needed to fill up your aquarium for animals to live. Assume that the aquarium has a rectangular shape that contains a cylindrical walkway to allow people to walkthrough the aquarium from one end to the other, your task is to create a program which takes in the dimensions of the aquarium and the pathway in meters to calculate for the volume of water needed to fill up the aquarium. (Note that the length of the aquarium is the same as the length of the walk way because it connects with both ends of the aquarium.) The formulae to calculate the volume of both shapes, the design of the aquarium, and the example of the interface of the program are as follow.

\[ \text{Volume of Rectangular Shape} = \text{Width} \times \text{Length} \times \text{Height} \]
\[ \text{Volume of Cylinder} = \pi \times \text{radius}^2 \times \text{Length} \]

```
Input the width of the aquarium: 22.5
Input the length of the aquarium: 325.8
Input the height of the aquarium: 7.6
Input the radius of the walkway: 1.3

The amount of water needed is 53982.033 cubic meters
```

9. Write a program to draw a board of 2x2 cells by using turtle module. User must enter the length of the board from keyboard and use this length to draw the board. Note that the turtle can start and stop at any points.

```
Length: 200
```
10. Find the biggest circle in the rectangle. Get the width and the height of rectangle from user. Print out the radius and the area of circle and use python module draw this circle at the middle of the rectangle. (Area of circle)

Enter the width of rectangle: 300
Enter the height of rectangle: 150

The radius of the biggest circle in this rectangle is 75.0
The area of the circle is 17671.46
Chapter 5

Conditional Structures

if

One of the very common situations when you write program is that you need the program to make decision for you. You may give it some conditions, and if the condition is true, you want it to do one thing or otherwise if the condition is false then do another thing. That’s the main topic for this chapter.

In Python we use a reserved word, if, for decision making. The usage pattern of this command is very simple. Just if follows by a condition and a colon :

```python
if condition:
    statement to execute if the condition is true.
```

If the given condition is true then the following indented statement will be executed, otherwise it will be skipped. See very simple example below:
In this example, the if will check whether x is actually more than 2. We saw the word “Superman!” because the condition is actually true but if we changed the value of x to 2 than nothing will be printed from this example because the condition is false.

You may want to print or execute more than one statement if the condition is true and you can do so by providing all that statements in the same indentation with the first one. The same indentation indicates that all of that statements are in the same “block of code”. With this, if the condition is true, the whole block will be executed but if the condition is false the whole block will also be skipped at once. So be very careful, indentation is very important in Python programming. The indentation is how Python knows that a particular block of code is separate from the code around it. The number of spaces used is important, and a Python-oriented programming editor will always carefully help you maintain the proper indentation for the code that is being written. The number of spaces is relevant, so it is important to use the editor to determine your indentation and not change the number of spaces manually.

```python
tax = 3
if x>2:
    print("Superman!")
Superman!
```

```python
tax = 3
if x>2:
    print("Superman")
    print("Supermom")
    print("Supergirl")
Superman
Supermom
Supergirl```

if..else

The else statement is an optional statement and there could be, at most, only one else statement following an if. An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a false value.

Following is the general form of a typical decision making structure found in most of the programming languages:
For the first example, the condition $x>2$ is true, so Superman was printed and Supermom was skipped. On the other hand, the condition $x>2$ in the second example is false so the whole block of print(“Superman”), print(“Supermom”) and print(“Superdad”) was skipped, and the Spiderman, Spidermom and Spiderdad are printed instead.

In case that there is just one statement following the if or the else, we can put that one statement on the same line of code with if or else for convenient and shorter code. However, if we want to have many statements following the if or else, then they have to be in a block of code with same indentation and cannot be on the same line with if or else.
If..elif..else

In case that you have many conditions or many cases to compare with, it’s very convenient to use special if..elif..else statement. The `elif` here actually is the short form of `else if`. Basically, we can use both of them in exchange.

```python
age = "16"
if age>10: print("Teenagers and above")
else: print("Kids")

Teenagers and above
```

Both examples are equivalent, but the first one looks a lot simpler thanks to the `elif` statement. One thing that we have to be careful when using `if..elif..else` is that the whole statement can be thought of as one big compound statement. It will start checking the first condition, the second, the third and so on until it finds a condition that is true, then it will process only the statements following that condition and skip the rest of the statement. It will not check anything in that big statement again. The effect can be clearly seen from this following examples (modified from previous example, just to show different style of programming to solve the same problem)

```python
buy=5500
if buy>10000: dis=10
elif buy>5000: dis=8
elif buy>1000: dis=7
else: discount=0
print("Get ",dis," % discount")

Get 8% discount
```

```python
buy=5500
if buy>10000: dis=10
else:
    if buy>5000: dis=8
    else:
        if buy>1000: dis=7
        else: discount=0
print("Get ",dis," % discount")

Get 8% discount
```

```python
x=5500
if x>10000:
    print("Discount is 10%")
elif x>5000:
    print("Discount is 8%")
elif x>1000:
    print("Discount is 7%")
else:
    print("No discount")

Discount is 8%
```

```python
x=5500
if x>10000:
    print("Discount is 10%")
if x>5000:
    print("Discount is 8%")
if x>1000:
    print("Discount is 7%")
else:
    print("No discount")

Discount is 8%
Discount is 7%
```
These two snippets look very similar, one with `elif` and one without. For the one on the left, when the condition \(x > 5000\) is found true, it displays the 8% sentence and skip the rest of the statement. For the one on the right, there are 3 `if`s and they are 3 different, detached statements. When the statement \(x > 5000\) is found true, it prints 8% sentence and finishes that statement. It then went on to the next statement, which is to check whether \(x > 1000\), and again this is true too, so Discount is 7% is also printed (unexpectedly, maybe). Just to make it clear, I’ll give you another similar example.

```python
print("This is a fortune teller program!")
print("1. Jasmine")
print("2. Rose")
print("3. Orchid")
c = int(input("Choose one out of 3 flowers here:"))
if c==1: print ("You are nice and clean")
elif c==2: print ("You are hot and sexy")
elif c==3: print ("You are exotic and fun")
else: print ("You are silly! I told you to choose from 1 to 3!!")
```

```python
print("This is a fortune teller program!")
print("1. Jasmine")
print("2. Rose")
print("3. Orchid")
c = int(input("Choose one out of 3 flowers here:"))
if c==1: print ("You are nice and clean")
if c==2: print ("You are hot and sexy")
if c==3: print ("You are exotic and fun")
else: print ("You are silly! I told you to choose from 1 to 3!!")
```

The only difference between these 2 snippets is the `elif` in the top one and the `if` in the bottom one. The first program will display only one sentence but the second program will always display 2 sentences no matter what number you choose. These two programs look similar but in fact they will produce different output. The above one will give the result as you expected but the 2nd example give you unexpected results.

**Note:** the `input()` function returns a string to the variable `c`, so `c` is a string variable. When you want to make a comparison to a string variable, you have to compare with “1” (a string) not 1 (a number). You can typecast it to integer with `int()` as shown in this program.

**Comparison operators**

Normally the expression that we use to test for the condition is the comparisons of somethings. Here are the list of the comparison operators that are allowed in the Python
ITS100 Programming is fun

Note that x=5 and x==5 are different! x=5 means we assign value 5 to x where x==5 means a question (a comparison) whether x is equal to 5 or not, if it is equal to 5 then it’s true or it will be false otherwise.

Note Be careful about the datatype! 5 is not equal to “5” because the first is a number where the second is a string. So they cannot be compared. If you want to compare them, you have to typecast one of them to be in the same type with the other.

Note Also be reminded that the input from the user is a string, so when you make a comparison, compare it with string, not a number. Or typecast it to a number first.

Logical Values

The result of comparison will be translated into just 2 values, True or False. For example 2==2 is equivalent to a True and 2>3 is equivalent to a False. Python has a special datatype dedicated to keeping the logical values called Boolean. So, if a type of a variable is Boolean that variable can either be True or False. You can assign this logical value directly to a variable with these 2 keywords, True and False. (Note that, they are case sensitive)

```
>>> a = True
>>> b = False
>>> c = true
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    c = true
NameError: name 'true' is not defined
```

The previous example shows that a variable a is a boolean variable with logical value True but there is an error assigning value true to c. The following example shows that we can directly assign a logical value to a variable and we can just test the variable in the if
statement. In this example, Superman is printed because the condition is true and Batman is also printed because the print("Batman") statement is not part of the if, so it will be printed anyway no matter condition is true or false.

```python
a = True
if a:
    print("Superman")
else:
    print("Spiderman")
print("Batman")

Superman
Batman
```

Python programming language assumes any non-zero and non-null values as True which means that any variable or value that a is not equal to zero can be used as True even though they are not a boolean variables. Either zero or null then it is assumed as False value. Anything that has a value, number of string, will be True too. In Python, integers and strings can be conditions of if statement. 0 and empty string are equivalent to False.

```python
x = 5
if x:
    print("Superman")
else:
    print("Spiderman")

Superman
```

The first example shows that a variable x can be used as a condition for the if statement, in this case x is not zero and Superman will be printed. For the second example, the result of y-x which is -3 is used for the condition for if and again, it's not zero so Superman will again be printed.

### Logical Operations

Normal decision making involves not just only one condition but many conditions. The conditions can get very complex and we need logical operations to determine the final outcome of the complex decision-making conditions.
Suppose that you have to check 2 conditions whether it's cold or not and whether or not it's raining. If it's really cold, but it's not raining, so the sentence “It's cold and it's raining” is then not true. Three other possibilities are shown in the table. The sentence can only be true if both “It's cold” and “It's raining” are true.

<table>
<thead>
<tr>
<th>It's cold</th>
<th>It's raining</th>
<th>It's cold and it's raining</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

And that's where we have to use truth table.

**Logical conjunction (and)**

Logical disjunction is an operation on two logical values, that produces a value of true if both of its operands are true.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X and Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

**Logical disjunction (or)**

Logical disjunction is an operation on two logical values, that produces a value of true if at least one of its operands is true.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X or Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
There are following logical operators supported by Python language. Assume that X is True and Y is False.

<table>
<thead>
<tr>
<th>operators</th>
<th>example</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>X and Y</td>
<td>FALSE</td>
</tr>
<tr>
<td>or</td>
<td>X or Y</td>
<td>TRUE</td>
</tr>
<tr>
<td>not</td>
<td>not (X or Y)</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Take a look a few examples. The first one is pretty simple, the condition is \((x>0) \text{ and } (y>0)\) and in this case, it's true, so Superman and Supergirl will be printed along with Batman which is not part of the if-else (it’ll be printed no matter the condition is true or false)

```python
x=5
y=2
if (x>0) and (y>0):
    print("Superman")
    print("Supergirl")
else:
    print("Spiderman")
    print("Spidergirl")
print("Batman")

Superman
Supergirl
Batman
```

The second example is similar, but this time we use conjunction, or. So if either of the condition \((\text{status}=="\text{student}"") \text{ or } (\text{age}<20)\) is true or both are true then the whole condition is true. In this case, the first one is true, so this student can not enter.

```python
status = "student"
age = 20
if (status =="student") or (age<20):
    print("No entry")
else:
    print("Welcome")
print ("It’s cool!")

No entry
It’s cool!
```

Last example is a bit more difficult. The condition is more complex. We will go to Japan based on 2 conditions, if we have enough money, says 100,000 Baht or if we have less than
ITS100 Programming is fun

40,000 Baht but Yen is not expensive, says 10 yens is less than 3 baht, then we will still be going.

```python
m=50000
ntenyen=4
if (m>100000) or ((m<40000) and (ntenyen < 3)):
    print("We will go Japan")
else:
    print("Eat ramen at home")
print ("and watch Batman")
eat ramen at home
and watch Batman
```

**Nested if**

In real situation, the we may have many conditions with complex relationship between them. We have to carefully translate them into programming language. And of course, there are normally more than one way to do that. Let’s have a look at this example where we want to classify students into many conference rooms based on the following conditions:

1. The 1st year male students go to room 209
2. The 1st year female students go to room 210
3. For the higher years, MT or EM students go to room 301
4. For the higher years, IT or CPE students go to room 302
5. For the higher years, other departments go to room 305

Let’s try to solve this problem in a straight forward manner. We have to ask each student who comes to the registration what year he is, and what department he is. And of course, we have to tell the program whether the student is a male or female. Three inputs are needed, then. We then need to form a list of conditions in the if..else pattern as in the following program.

This program works well and correct but you may notice that the conditions are complex and prone to error.
ITS100 Programming is fun

However, with many complex conditions, a small mistake in one of the conditions, such as missing of ( ) on the 3rd or 4th condition, can lead to wrong result and that mistake, if happened, will be difficult to detect. Such a mistake is what we called a “logical error”. There is nothing wrong with the code, no grammatical error, so the editor or the parser can’t detect it. Just the logic that is wrong.

We can avoid such mistake by simplify the conditions with structured, nested if..else. The following code will produce the same result as the previous one but conditions are simpler. We first check if the student is a first year or not, if yes then we start a new block of code that takes input of the sex and just classify student easily with simpler condition. On the other hand, if he is not a first year, we start a new block of code. Within that block, we check his department and classify them into rooms with simpler conditions too.

```
year = int(input("What year are you?: "))
sex = input("male or female?: ")
dept = input("What department are you?:")
if year==1 and sex == "male": room = 209
elif year==1 and sex == "male": room = 210
elif year>1 and (dept=="MT" or dept=="EM"): room = 301
elif year>1 and (dept=="IT" or dept=="CPE"): room = 302
else: room = 305
print ("Your room is:", room)
```

These 2 different styles have different advantages and disadvantages. The first one is simpler in terms of programming structure. It looks very linear and simple to write. However, the second one is simpler in terms of conditions checking. The conditions are clear and easy to understand, but the code may be more complex. So you have to choose the style for the appropriate problem.
The following example shows very complex flowchart of the problem. It may be more appropriate to use nested if structure than the linear one.

![Flowchart Image]

**Conclusions**

We have done the condition checking in this chapter. The command is so simple just **if** or **if. else** and **if. elif. else** and you have seen that we can use this simple command in a variety of ways to solve many problems. No matter how complex the problem is, they can be solved with these simple **if. else** structure. As I told you, programming is more of an art than a science. One simple command and if you master it, you can use it to solve many complex problem. Just like playing chess, we can learn to move each piece easily, but to be a good chess player we need a lot of practice. The program can get more and more complex when the conditions or logic get complex. So keep the basic right. **if** follows by a condition or conditions, then a colon : There will be many more problems of “**if**” in the exercise sections.
Exercises

1. **Coffee shop**: Your friend owns a coffee shop. In his cafe, a normal price for a cup of coffee is 40 Baht. However, if a customer wants stronger coffee, customer can request as many extra shots as they like with the cost of 15 Baht per extra shot. In his shop, he also has many healthy options for his customer, like offering soy milk and natural honey. If the customer requests for soy milk, another 10 baht will be added. If customer wants to have natural honey instead of normal syrup, customer will have to pay another 20 Baht extra. That’s all options he has for a cup of coffee. OK, now, he would like us to write a program to help his barista to calculate the price for each cup of coffee.

2. **Sorting Hat**: It’s tradition when you come to Hogwarts School of Witchcraft and Wizardry, you will be sorted by the Sorting Hat into 1 of the 4 Houses and they are Gryffindor, Hufflepuff, Ravenclaw and Slytherin. Rumor has it that in fact the Sorting Hat is actually not magical but simply a Python program written by Professor Albus Dumbledore during his 1st year at SIIT. He revealed his secret in the 8th *Harry Potter and The Turtle Python* that the sorting hat will just ask for the name of the person who is wearing it. If the name of that person begins with **H**, **B** or **N** (like Harry Potter, Hermione Granger or Neville Longbottom), it will put the person in Gryffindor. If the person’s name begins with **E** or **C** (like Elephant or Cedric Diggory), the person will be Hufflepuff. The person will be added to Ravenclaw house only if the name begins with **G** or **L** (like Gilderoy Lockhart or Luna Lovegood). The rest of the name will be in Slytherin. Problem is the Sorting Hat has been stolen from Hogwarts, so Prof. Dumbledore ask you to write the new one for him.

3. **Car payment**: Suppose that your father is the Toyota car dealer and there are 3 different types of Toyota in his showroom.
   a. Toyota Camry costs 1,700,000 Baht
   b. Toyota Altis costs 900,000 Baht
   c. Toyota Vios costs 559,000 Baht

   Every time when customers come and ask for the details of payment, a salesperson asks customer what type of Toyota the customer wants, a, b, or c. He then asks for how much the customer want to put for Down payment. Then how many months is customer willing to pay? Then the salesperson has to calculate the amount of payment the customer has to pay each month but the whole process is slow and has lots of mistakes. Actually, it can easily be calculated by
   
   \[ \text{each month payment} = \frac{(\text{Car price} - \text{Down payment})}{\text{number of months}} \]

   Can you write a program for your father so everything will be faster and correct?
4. **Abstract body shape:** Ask a user this question, “How old are you?” so that the program can predict the future shape for the user. If the user’s age is in between 1 - 10 then use the Turtle to draw 2 small rectangle. If the age is more than 10 but not more than 20, the draw a big red triangle. If more than 20 but less than 30 then draw a snow man with 3 circles. If more than 30, the draw a big fat blue circle.

5. Write a program that takes three integer numbers and show the maximum number. Only use if( ) for comparison. Additional functions are not allowed.

   ```
   Enter three integer numbers: 5 16 11
   The maximum number is 16.
   ```

6. Write a program that authenticates a user by asking for her passcode. If she input a word “Secret” then grant her an access, otherwise deny her access.

   ```
   Enter your passcode: John
   Access Denied!
   
Enter your passcode: Secret
   Access Granted!
   ```

7. Write a program that checks whether an input integer is negative or positive.

   ```
   Enter an integer number: -1
   The number you enter is negative.
   
Enter an integer number: 10
   The number you enter is positive.
   ```

8. Write a program which reads two integer values. If the first number is less than the second number then print the message “Up!”. If the second number is less than the first number, print the message “Down!”. If the numbers are equal, print the message “Equal!”.

   ```
   Enter three integer numbers: 5 16 11
   The maximum number is 16.
   ```

   ```
   Enter your passcode: John
   Access Denied!
   
Enter your passcode: Secret
   Access Granted!
   ```

   ```
   Enter an integer number: -1
   The number you enter is negative.
   
Enter an integer number: 10
   The number you enter is positive.
   ```
9. Write a program that translates the meaning of traffic light colors. Red light is represented by character ‘R’ and green by ‘G’. If the user inputs ‘R’, the program will print out “Stop”. If the user inputs ‘G’, the program will display “Go!”.

```
Input traffic light color (R=Red,G=Green): R
Stop!
```

10. Write a program that gets five integer values and shows the maximum and minimum values among them.

```
Input1#:0
Input2#:−7
Input3#:3
Input4#:11
Input5#:4
Maximum: 11
Minimum: −7
```

11. Write a program that takes 3 characters and then counts the number of consonants and vowels as shown in the following example.

```
Input three characters:> z H a
The number of consonants is 2.
The number of vowels is 1.
```

12. The Lucky numbers for this month are 8, 12, 20, 55. Write a program to get 3 numbers from user if at least 2 of the numbers match with the lucky numbers then print, “You won!” otherwise say “You lose”

```
Input 1: 20
Input 2: 22
Input 3: 18
You lose!
```

```
Input 1: 15
Input 2: 8
Input 3: 55
You won!
```
13. Write a program that gets a size of octagon's side. If the size of side is less than 100 draw the orange octagon otherwise draw the black colour octagon. (Octagon can be created of lines and turns - at the appropriate angle)

Enter the size of octagon: 50

14. Write a program that takes a colour from user and draw a heart with that colour.

What is the colour of you heart?: red

What is the colour of you heart?: blue

15. Ask the user “What day is today?” Give her an angry face if she gives a wrong answer.
Chapter 6

Repetition Structure

Repetition

The next useful concept for programming is a repetition structure.

When you write a bigger program or solve a more complex problem, it is very usual that you need to repeatedly execute a block of code several times in the program. In general, statements are executed sequentially: The first statement in is executed first, followed by the second, and so on. Programming languages, such as Python, provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. In Python, two commands, for and while, are very common and very useful in the repetition structure. For loops are normally used when you have a block of code that you want to repeat n number of times (normally, n is known). As an alternative, there is the while loop. while is normally used to repeat a block of code until a certain condition is met, or if you want the block of code to repeat forever, for example.

Let’s begin with the for loop.
For loop

A `for` loop is a Python statement which repeats a block of code a specified number of times. It has the ability to iterate over the items of any sequence such as strings, arrays, lists, tuples, dict and so on. (We will only talk about a list in this chapter, the rest will be discussed in the following chapters.) The syntax of a for loop look is as follows:

```
for x in sequence:
    statement
    statement
```

A `for` follows by an iterating variable and a word `in`, then a sequence. So the pattern is simply `for .. in ..`. If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable `x`. Next, the statements block is executed. Next item in the list will be assigned to `x`, and the statement block is executed again and it keeps on doing this until the entire sequence is exhausted. The following diagram shows the flow of the command.

OK, let's learn from two very simple examples here.

```
for x in (1,2,3):
    print("Superman")
Superman
Superman
Superman
```

```
for x in (1,1,1):
    print("Superman")
Superman
Superman
Superman
```
In the left example, we let the variable \( x \) iterates over a list of \((1,2,3)\). So in the first round of the `for` command, \( x \) is assigned with the 1st element of the list, which is 1. It then prints a Superman, which is the only command to be repeated. After the first round is finished, the second number in the list is assigned to the variable \( x \) and this time \( x \) is 2. And again it prints Superman. It keeps on doing this until the last item in the list is assigned and finished. The example on the right will be just the same, it repeated 3 times, printing 3 Supermans. The have the same results because there are 3 items in the list and variable \( x \) is not involved in any calculation or printing. If \( x \) is a part of the print out, the results will look like this.

\[
\begin{array}{|c|}
\hline
\text{for } \text{x in } (1,2,3): \\
\quad \text{print(\text{x, " Superman")} \\
1 \text{ Superman} \\
2 \text{ Superman} \\
3 \text{ Superman} \\
\hline
\end{array}
\]

And you can notice that the value of \( x \) depends on the items in the list. Of course, we can use that variable \( x \) (or other variable name) in any calculation we may want. Let’s print a multiplication table with 2 lines of code.

\[
\begin{array}{|c|}
\hline
\text{for } \text{n in } (0,1,2,3,4,5,6): \\
\quad \text{print(\text{n,"* 2 =",n*2)}} \\
0 \times 2 = 0 \\
1 \times 2 = 2 \\
2 \times 2 = 4 \\
3 \times 2 = 6 \\
4 \times 2 = 8 \\
5 \times 2 = 10 \\
6 \times 2 = 12 \\
\hline
\end{array}
\]

With this example, \( n \) is the iterating variable and a list containing number 0 to 6. In the first round, the first element of the list, 0, is assigned to variable \( n \). We then print value of \( n \) follow by \(* \ 2 =\) and the value of \( n \) multiplied by 2. So we have \( 0 \times 2 = 0 \) for the first line. The first iteration is finished. The second iteration is started. The second number in the list, 1, is then assigned to \( n \) this time. So it will display \( 1 \times 2 = 2 \). It will keep doing this until the last element of the list, 6, is assigned and printed \( 6 \times 2 = 12 \).

Same idea for a block of code is also applied here. If a block of code needs to be repeated, we need to put them with the same indentation. In the following example, Superman and Supergirl are printed twice because there are 2 items (10 and 20) in the list. When the first item, 10, is assigned to a variable counter during the first round, it prints 10 in front of
Superman and it print counter+1, 11, in front of Supergirl. Batman is not part of the loop (notice the indent) so it is printed only once.

```python
for counter in (10,20):
    print(counter,"Superman")
    print(counter+1,"Supergirl")
print("Batman")
```

10 Superman
11 Supergirl
20 Superman
21 Supergirl
Batman

The list can be not only the numbers, but also something else. The following example show that we can put names in a list and then used it in for loop. Each member of the list will be assigned to the variable. The command will just fetch each member and give the value to intreating variable s and then start to process the statement (or statements) below. When all the statements under the for command are all processed, that's the end of each iteration. The new iteration starts by fetching the next value in the range's list and give it to s, and process the statement again with the new value s. And the program keeps doing this until all the members in the range has been exhausted.

```python
for s in ("Superman", "Supergirl", "Supermom"):
    print(s,"is part of the Super family")
print("Batman is not")
```

Superman is part of the Super family
Supergirl is part of the Super family
Supermom is part of the Super family
Batman is not

It’s convenient to list all the numbers in a sequence and provide it to the for command. However, some times, we can not know all that numbers in advance. Or maybe the list is too long, like 1 to 10000, so it’s cumbersome to list them all in this manner. There is another command that helps with this problem.

**Range**

If you do need to iterate over a sequence of numbers, the built-in function range (**start, stop, step**) comes in handy. It generates lists containing arithmetic progressions. Three parameters, start, stop and step, are needed. It generates a list of numbers starting with
start and ends just before stop value with increment step. The given end point is never part of the generated list.

The first example, range(0,10,1) is to generate a list of numbers, starting from 0 and stop just before 10 and increase by 1. The second example has the same start point and stop point but the increment is 3 so the last number is 9. We can set the incremental step with a negative number as shown in the last example, where starting point is -10 and end point is -100.

```python
>>> range(0,11,1)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 11, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

If the conditions given to the range are not appropriate and the list couldn’t be generated then it will return an empty lists as shown in the following examples.

```python
>>> range(0,-5,1)
[]
>>> range(-20, -11, -3)
[]
>>> range(-10, 10, -2)
[]
```

**Range (omitting Step)**

You can omit a few parameters for the range command. If step is omitted, it will be 1 by default. So range (start, stop) means range (start, stop, 1)

```python
range(start, stop) means range (start, stop, 1)
```

These 3 following examples show that when there are only 2 parameters provided to the range command, that 2 parameters are considered to be start and stop and the value of step will be set to 1 by default.
Range (omitting Start and Step)

A start can also be omitted. When you provide just only 1 parameter to the `range` command, it will be considered as Stop and set Start to 0 by default and again Step will be set to 1 by default.

```
range(stop) means range (0, stop, 1)
```

the range will start from 0. So `range (stop)` means `range (0, stop, 1)`

```python
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(-10)
[]
>>> range(8)
[0, 1, 2, 3, 4, 5, 6, 7]
```

Note that `range` command doesn’t accept floating point and string parameters.

```python
>>> range(0.1, 0.5, 0.1)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    list(range(0.1, 0.5, 0.1))
TypeError: 'float' object cannot be interpreted as an integer

>>> list(range("a","c",1))
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    list(range("a","c",1))
TypeError: 'str' object cannot be interpreted as an integer
```
For in Range

Now it's more convenient that we combine the two functions, `for` and `range`. We can generate a list with the range command and give each value to the iterating variable.

```
for x in range(a, b, c):
    statement
```

Have a look at the following example. With only 2 lines of code, we can generate some useful results. In this example, `range(10)` is translated into `[0,1,2,3,4,5,6,7,8,9]` before the for command fetches each element from that list.

```
for x in range(10):
    print(x, "squared equals", x*x)

0 squared equals 0
1 squared equals 1
2 squared equals 4
3 squared equals 9
4 squared equals 16
5 squared equals 25
6 squared equals 36
7 squared equals 49
8 squared equals 64
9 squared equals 81
```

If we want to calculate just only the square of even numbers, we can easily modified the range command so that it generates only the even number. The rest of the program is unchanged.

```
for x in range(0, 10, 2):
    print(x, "squared equals", x*x)

0 squared equals 0
2 squared equals 4
4 squared equals 16
6 squared equals 36
8 squared equals 64
```
Nested-For

For complex calculation, we may need more than 1 iterating variable to solve it. The for command can reside within another for. That’s another useful concept and that always cause confusion for beginners. Consider this example:

```
for x in range(4):
    for y in range(3):
        print(x, y)
```

We have 2 loops here, one inside another. Let’s call it loop x and loop y. Loop x will repeat 4 times and the value of x will be changing from 0, 1, 2 and 3. In each iteration of loop x, we have another loop, loop y, inside it. Loop y repeats 3 times changing values from 0, 1 and 2. So for each iteration of loop x, the command in loop y will run 3 times. Thus, the result 12 lines, not 7 lines.

The following example shows a table generated from a nested loop, one loop inside another. For the loop y, we print x*y followed by a tab indicated by \t. Once loop y is done, we just call a normal print() which automatically display a new line.

```
for x in range(1,6):
    for y in range(10,60,10):
        print(x*y,end="\t")
    print()
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>60</td>
<td>80</td>
<td>100</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>90</td>
<td>120</td>
<td>150</td>
</tr>
<tr>
<td>40</td>
<td>80</td>
<td>120</td>
<td>160</td>
<td>200</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
<td>250</td>
</tr>
</tbody>
</table>
While Loop

A **while** loop statement in Python programming language repeatedly executes target statement as long as a given condition is true. The syntax of a while loop in Python programming language is:

```
while condition:
    statement
```

Here statement may be a single statement or a block of statements (Remember, in Python, all the statements indented by the same number of character spaces are considered to be part of a single block of code. Python uses indentation as its method of grouping statements). The condition may be any logical expression or logical value (non zero value is considered to be true). The loop keep on iterating while the condition is true. When the condition becomes false, program will skip the statement and go to the line immediately following the loop. This means that if the condition is always true, the loop will run infinitely. Diagram below show the flowchart of the while loop.
The following example shows simple structure of the while loop. The condition \(5 > 4\) is always true, so the word “Superman” will be printed infinitely.

```python
while 5 > 4:
    print("superman")
```

In the next example, \(x\) is assigned to 5 at the beginning, so a condition \(x > 0\) is true and the superman will be printed, then \(x\) is reduced by 1 for each iteration. Next round \(x\) is 4 and it will still be true, Superman will be printed. It keeps going like this until \(x\) is 0, that’s where the condition is false. So Superman will be printed 5 times from

```python
x = 5
while x > 0:
    print("Superman")
    x = x - 1
```

Superman
Superman
Superman
Superman
Superman
Extra material: Random

We have seen 2 important modules so far, and they were Math module and Turtle module. Random module is another useful one and is going to be introduced in this chapter. Same as other modules, we can get all the functionalities in the module into our program by importing it with import command.

import random

There are many useful commands in this module but one of the very useful that I would like to introduce here is a function .randint(a,b) The function will return a random integer number that are in between a and b, inclusively.

By calling x = random.randint(1, 100), x will be a random number between 1 - 100. Try to run this piece of code several times and you will see that each time you get different numbers.

```
import random as r
while x>10:
    x = r.randint(1,100)
print (x)
```

Random number is very useful when you want your program to come up with unpredictable results. The result that even the programmer doesn’t know the outcome of the program. It could be used in games or puzzles.

Extra Material - if .. in range

Not only for command that can be used in combination with range, the if command has that ability too. This following example shows that we can check whether the value of a variable is a member of a given list or not.
Conclusions

We have discussed about the simple repetition structure using `for` loop and in combination with `range` together to produce the important tool for Python programming.

```python
x = input("Give me a character: ")
if x in ('a', 'e', 'i', 'o', 'u'):
    print(x,"is a vowel")
```
Exercises

1. Fill the corresponding pair of range commands and results.

<table>
<thead>
<tr>
<th>range</th>
<th>all elements in range</th>
</tr>
</thead>
<tbody>
<tr>
<td>range (0, 10, 1)</td>
<td></td>
</tr>
<tr>
<td>range (0, 10)</td>
<td></td>
</tr>
<tr>
<td>range (10)</td>
<td></td>
</tr>
<tr>
<td>range (1, 40, 5)</td>
<td></td>
</tr>
<tr>
<td>range (-2, 20, 4)</td>
<td></td>
</tr>
<tr>
<td>range (12, 1, -2)</td>
<td></td>
</tr>
<tr>
<td>range (-10, 10, 3)</td>
<td></td>
</tr>
<tr>
<td>range (10, 3)</td>
<td></td>
</tr>
<tr>
<td>range (0, 10, -2)</td>
<td></td>
</tr>
</tbody>
</table>

2. **Shell:** Write a program that asks for a number of circles the user wants to draw, then draw the circles similar to the one displayed below.

![Shell diagram]

3. **Square:** (a) Write a program to calculate square of \( n \) numbers where \( n \) is the input from user and (b) just display only the square of even numbers.

4. **Bank:** A bank needs a program that help customers to calculate their account’s balance. The program should ask customer for the amount the customer wants to deposit and how many years they want to calculate. The program assumes that the interest rate is 5% per year and display the projected balance of each year for customer.
5. **Candy:** Write a program to draw a candy (as displayed here). The candy picture here is made of 100 circles. The size of the circle is reduced from 100 to 1. The colour of the circle will be alternatively changed from red, pink, yellow, green and blue. (Hint: it’s combination of if and for)

6. Get n from user and draw a flower with n petals.
   Draw each petal with
   ```python
turtle.circle(100,90)
turtle.left(90)
turtle.circle(100,90)
```
   Also change the color of each petal too (hint: if within for and %(mod) operator
   (n is 12 for this example)

7. **Candy:** Write a program to draw a candy (as displayed here). The candy picture here is made of 100 circles. The size of the circle is reduced from 100 to 1. The colour of the circle will be alternatively changed from red, pink, yellow, green and blue. (Hint: it’s
combination of if and for)

8. Write a program to generate bubbles on a canvas size of 1000x800 pixels at random location, random sizes and random colour. Repeat the program until one bubble falls below -350 (on the X axis)
9. Later change these bubbles into flowers with random sizes and colors too. (This
demonstrate the loop within the loop concept.)