

## Contents

Uses Problem Solving Process .....	3
Steps of the Process .....	3
Top Down Design and Stepwise Refinement Methodologies.....	3
Modularization .....	3
Structure Charts.....	4
What is a structure chart?.....	4
Notation.....	4
Sample Structure Chart 2 .....	8
Sample Structure Chart 3 .....	9
Summary – Structure Charts Consist of .....	9
Algorithms.....	10
Flow Chart – Symbols.....	10
Pseudocode .....	10
Programming Paradigms .....	11
What is a Programming Language? .....	11
Characteristics of a programming language.....	11
Evolution of Programming Languages.....	11
Programming Paradigms.....	11
Program Translation .....	12
Integrated Development Environment .....	13
Python IDLE .....	13
Python.....	14
Identifiers.....	14
Reserved Words.....	14
Lines and Indentation .....	15
Quotation in Python.....	15
Comments in Python.....	15
Variables .....	16
Assigning Values to Variables .....	16
Multiple Assignments.....	16

Data Types .....	17
Strings .....	17
Lists .....	18
Tuples .....	19
Dictionary .....	20
Type Conversions .....	21
Operators .....	21
Types of Operators .....	21
Arithmetic Operators .....	21
Comparison Operators .....	22
Assignment Operators .....	23
Bitwise Operators .....	24
Logical Operators .....	25
Membership Operators .....	25
Python Operators Precedence .....	25
Decision Making .....	26
Single Statements .....	26
else if OR elif .....	27
Loops – Iteration .....	28
While Loop and For Loop .....	28
Intelligent for loop .....	29
Nested Loop .....	30
Loop Control Statements .....	31
Sub Programs – Functions .....	32
File Handling .....	33
Searching and Sorting .....	34
Searching Technique: Sequential Search .....	34
Sorting Technique: Bubble Sort .....	35



## Uses Problem Solving Process

### Steps of the Process

1. Understanding the problem

..... Identify the problem .....

2. Defining the problem and boundaries

..... Analyzing the problem .....

3. Planning solution

..... Designing a solution .....

4. Implementation

..... Developing the solution .....

Sub division of a System

### Top Down Design and Stepwise Refinement Methodologies

A top-down approach (also known as stepwise design) is essentially the breaking down of a system to gain insight into the sub-systems that make it up. Stepwise Refinement. A way of developing a computer program by first describing general functions, then breaking each function down into details which are refined in successive steps until the whole program is fully defined. Top-down design involves looking at the whole task and breaking it down into smaller, more manageable sub-problems which are easier to solve.

### Modularization

The process of breaking the problem into smaller parts, and implementing each part separately is called modularization. Modular programming is the process of subdividing a computer program into separate sub-programs. A module is a separate software component.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

## Structure Charts

### What is a structure chart?

It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail. In modular programming, the complete system is coded as small independent interacting modules. Each module is aimed at doing one specific task. The design for these modules is prepared in the form of structure charts.

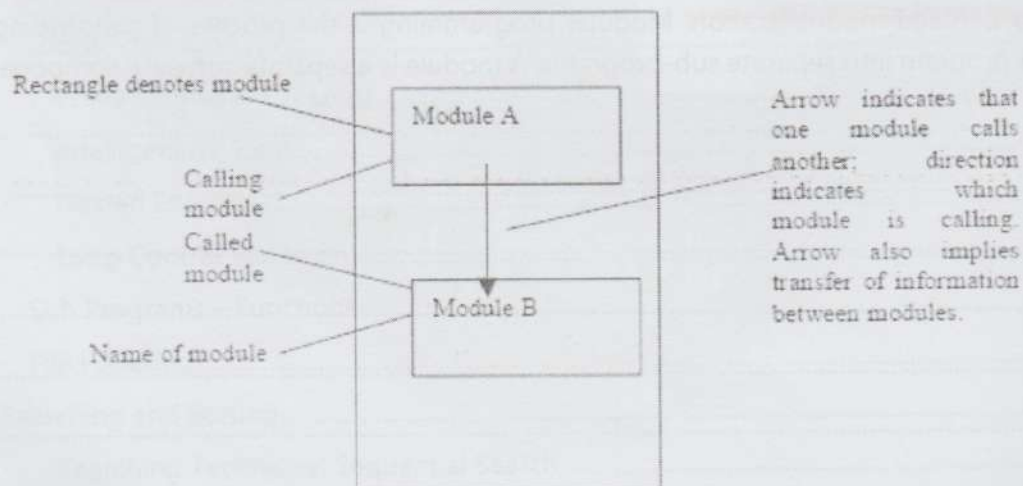
Structure chart represents hierarchical structure of modules. At each layer a specific task is performed. Structure charts show how variables pass between modules in a computer program. They show the hierarchy of module calls.

Structure charts show the relation of processing modules in computer software. It is a design tool that visually displays the relationships between program modules.

Structure charts are developed prior to the writing of program code. They identify the data passes existing between individual modules that interact with one another.

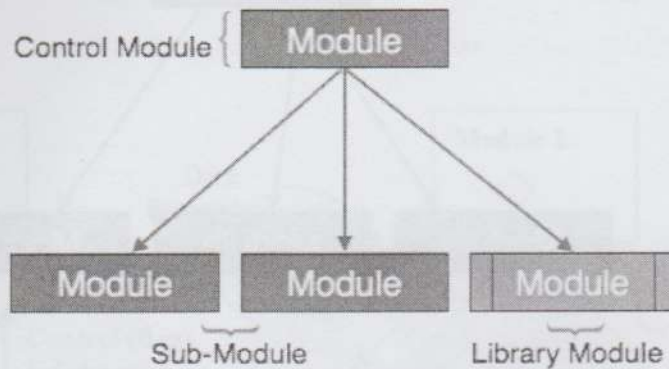
### Notation

Program modules are identified by rectangles with the module name written inside the rectangle. Arrows indicate calls, which are any mechanism used to invoke a particular module.

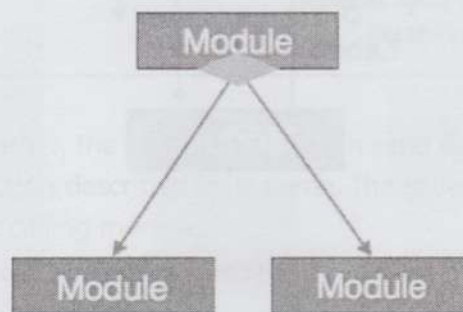


Here are the symbols used in construction of structure charts –

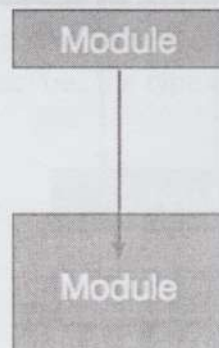
**Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



**Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.

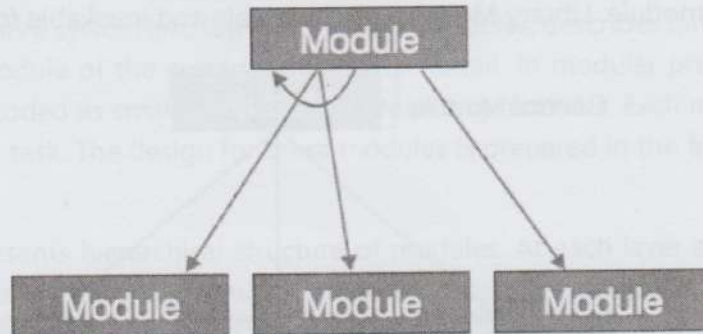


**Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.

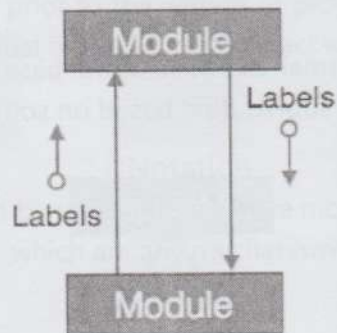




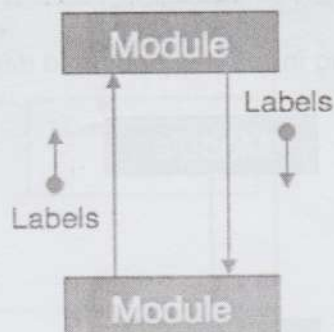
**Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



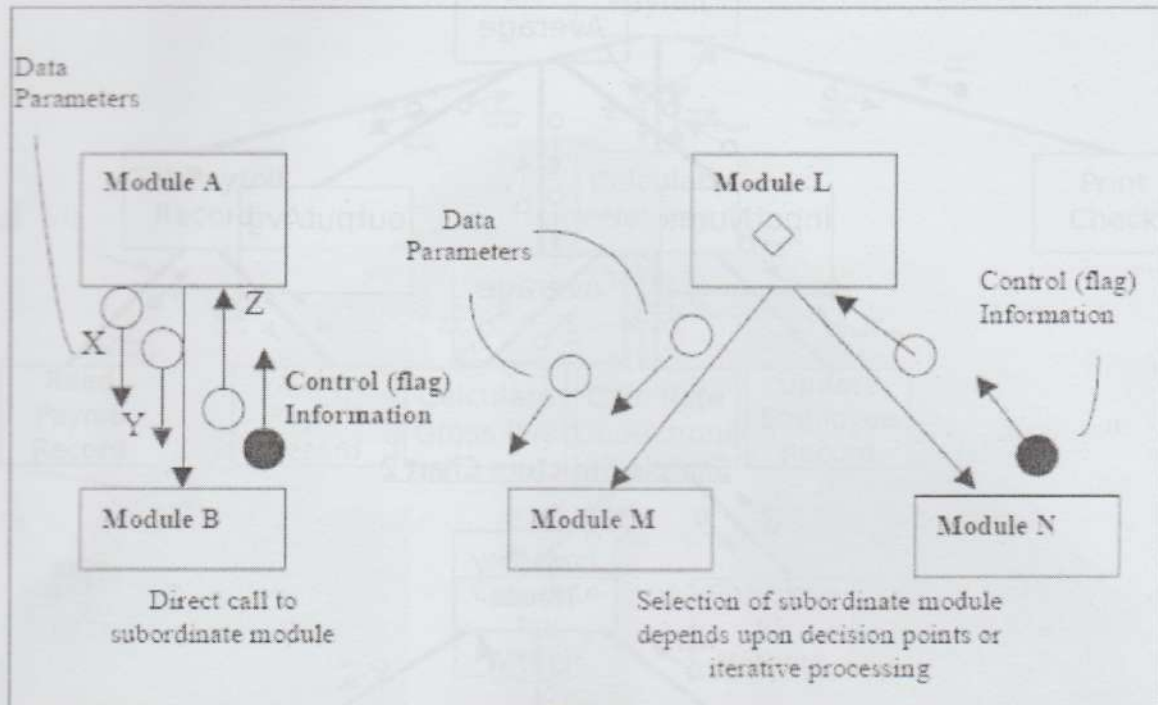
**Data flow** - A directed arrow with empty circle at the end represents data flow.



**Control flow** - A directed arrow with filled circle at the end represents control flow.



A calling module can interact with more than one subordinate module. The following figure also shows module L calling subordinate modules M and N. M is called on the basis of a decision point in L (indicated by the diamond notation).



When one module calls another, the calling module can send data to the called module so that it can perform the function described in its name. The called module can produce data that are passed back to the calling module.

Two types of data are transmitted. The first, parameter data, are items of data needed in the called module to perform the necessary work. A small arrow with an open circle at the end is used to note the passing of data parameters. In addition, control information (flag data) is also passed. Its purpose is to assist in the control of processing by indicating the occurrence of, say, errors or end-of-conditions. A small arrow with a closed circle indicates the control information. A brief annotation describes the type of information passed.

.....

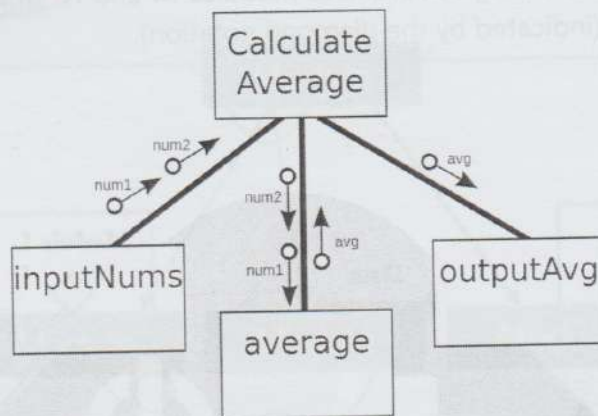
.....

.....

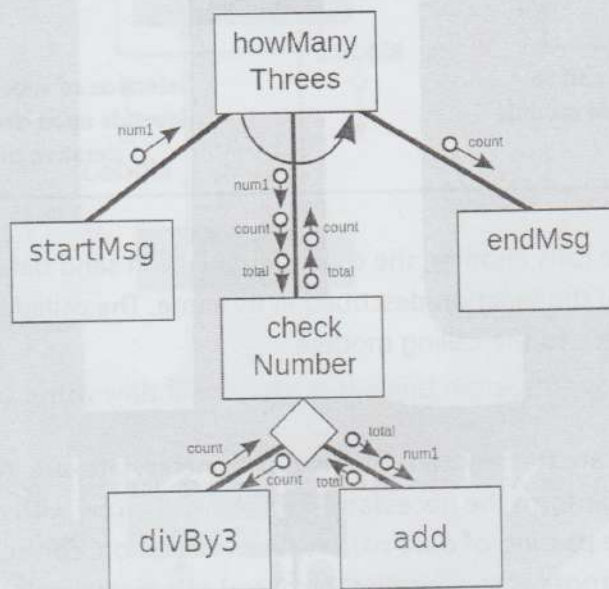
.....



Sample Structure Chart 1

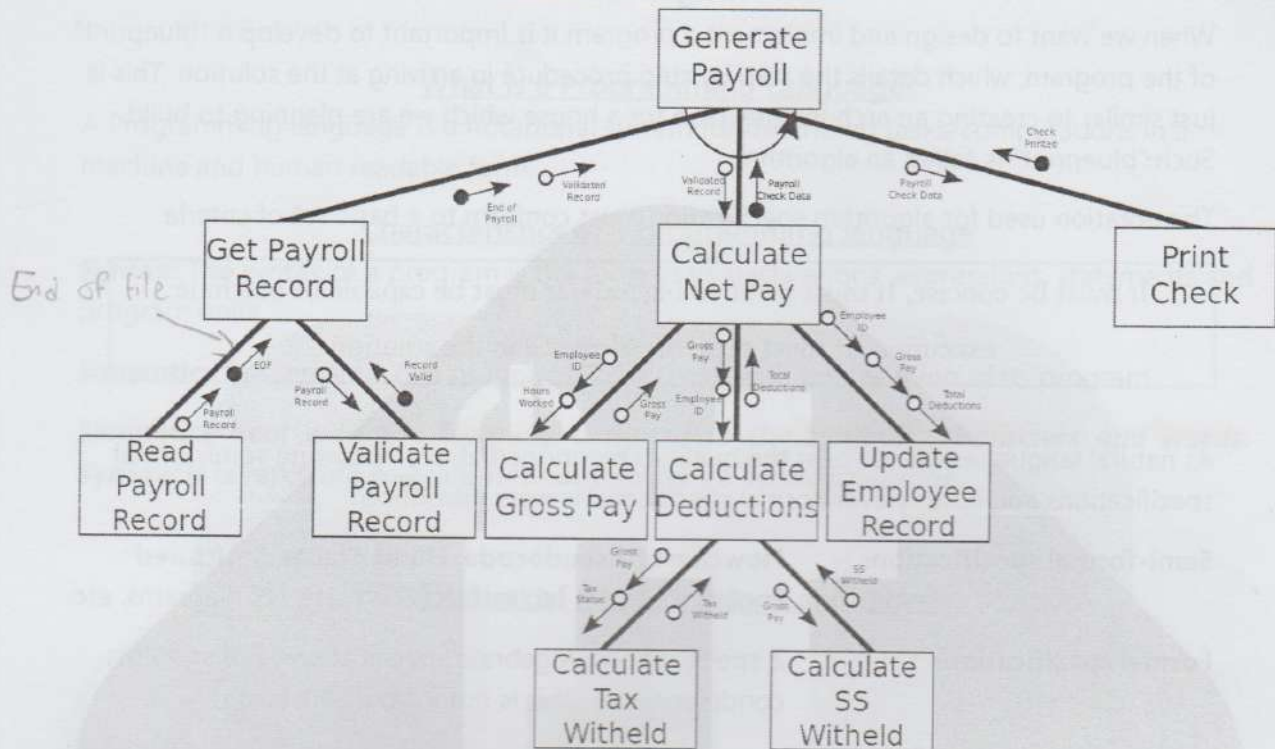


Sample Structure Chart 2





Sample Structure Chart 3



Summary – Structure Charts Consist of

Modules - Related program code organized into small units that are easy to understand and maintain

Data couples - Data passed between modules

Control couples - Data passed between modules that indicates a condition or action to another module (e.g. End of File)

Conditions - determines what subordinate module a control module will run

Loops - Indicates one or more subordinate modules are repeated

## Algorithms

When we want to design and implement a program it is important to develop a "blueprint" of the program, which details the step by step procedure in arriving at the solution. This is just similar to creating an architectural plan for a house which we are planning to build. Such "blueprint" is called an algorithm

The notation used for algorithm specification must conform to a basic set of criteria

It must be concise, It must be unambiguous, It must be capable of machine execution, It must promote elegance in the solution

As natural languages do not bear the qualities mentioned above, following semi-formal specifications and latterly even formal specifications were introduced.

- Semi-formal specifications** – **Flowcharts, Pseudocode, Hand Traces, Structured English, Decision tables, Decision trees, NS diagrams, etc.**
- Formal specifications** – Z specifications, Algebraic Specifications, Pre and Post conditions, etc. (this is not in your syllabus)

### Flow Chart – Symbols

### Pseudocode

## Programming Paradigms

### What is a Programming Language?

A Programming language is a notational system for describing tasks/computations in a machine and human readable form.

### Characteristics of a programming language

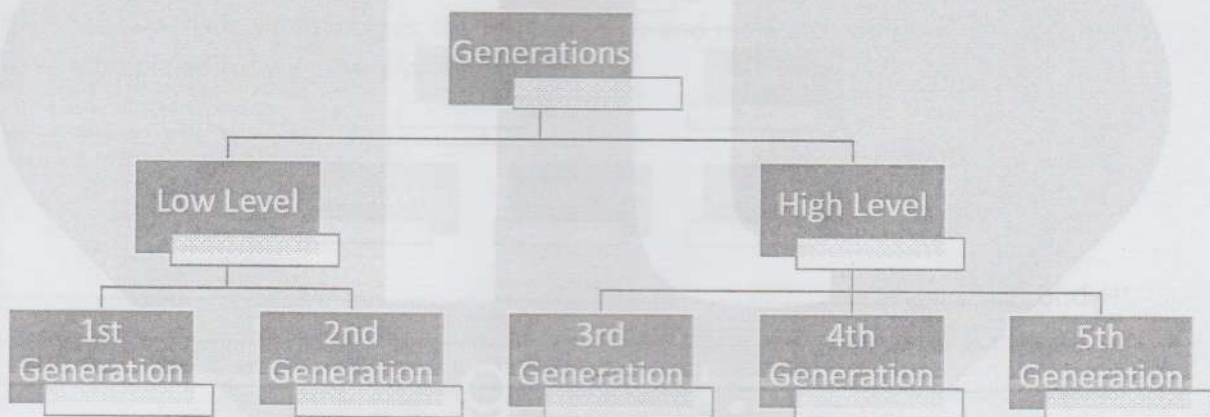
**Syntax:** The syntax of a program is the form of its declarations, expressions, statements and program units.

**Semantic:** The semantic of a program is concerned with the meaning of its program

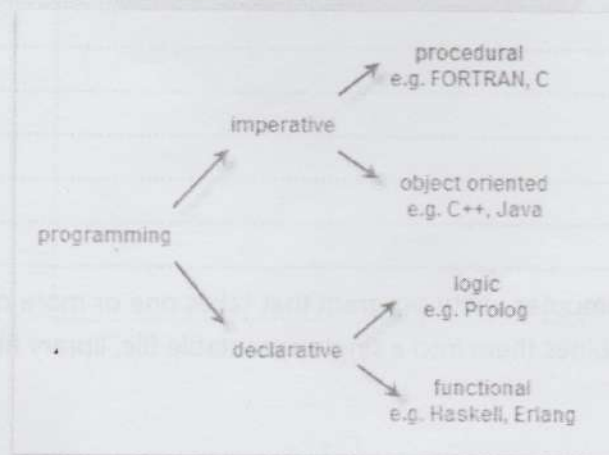
*Semantics deal with the meaning assigned to the symbols, characters and words.*

*Syntax: It is referring to grammatically structure of the language*

### Evolution of Programming Languages



### Programming Paradigms

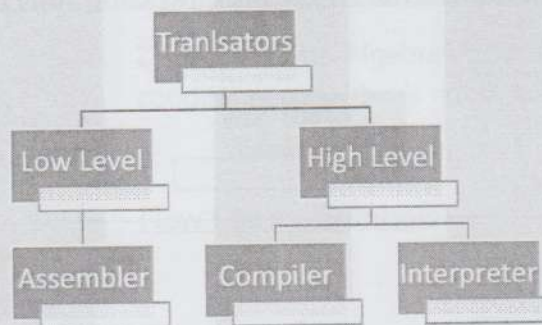




**Declarative** - Declarative programming is a computer programming paradigm that the developer defines what the program should accomplish rather than explicitly defining how it should go about doing so. Ex: SQL, Prolog

## Program Translation

the source code into a machine



**Linker** - A linker is a computer utility program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another 'object' file.

## Integrated Development Environment

An integrated development environment (IDE) is a software suite that consolidates the basic tools developers need to write and test software. Typically, an IDE contains a code editor, a compiler or interpreter and a debugger that the developer accesses through a single graphical user interface (GUI).

---

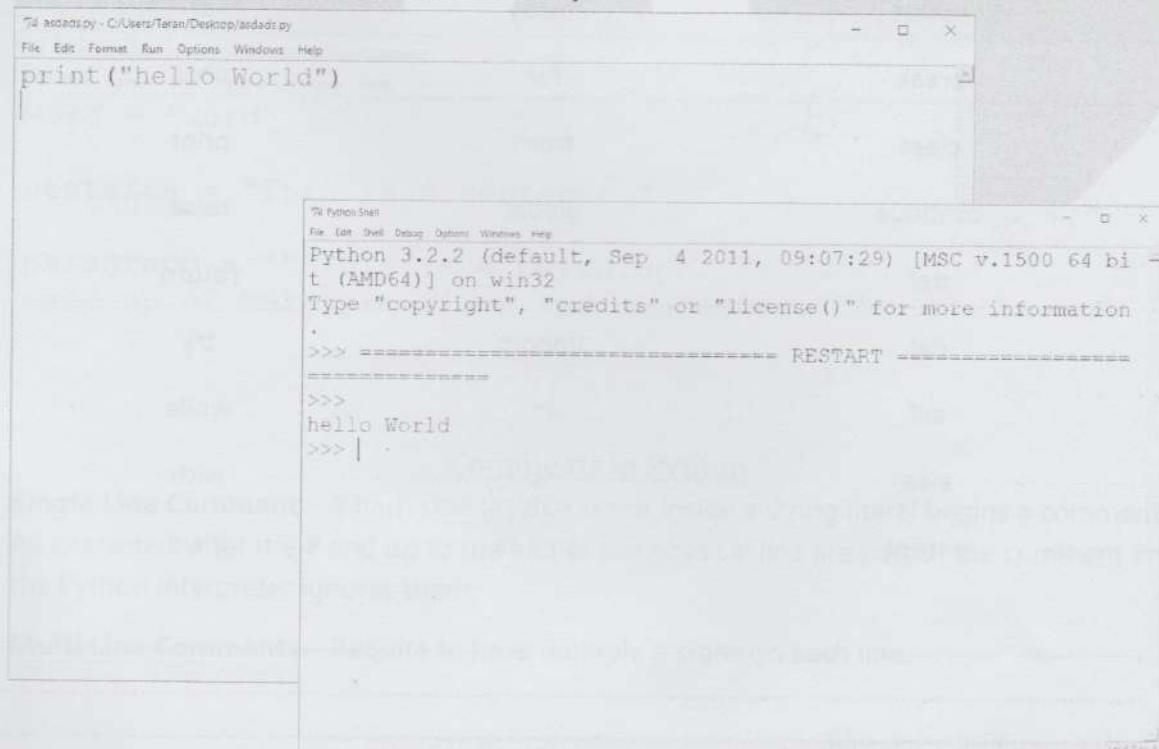
### Python IDLE

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the tkinter GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously.



## Python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

### Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).
- Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

### Reserved Words

These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

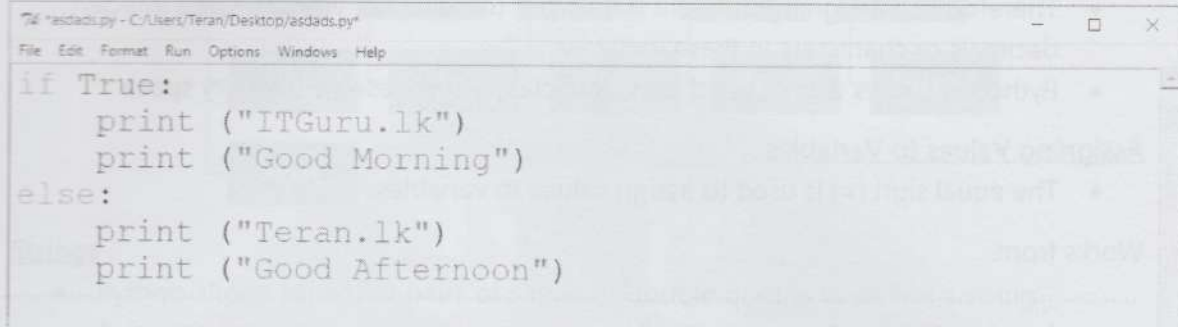
and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield



### Lines and Indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.

Ex:

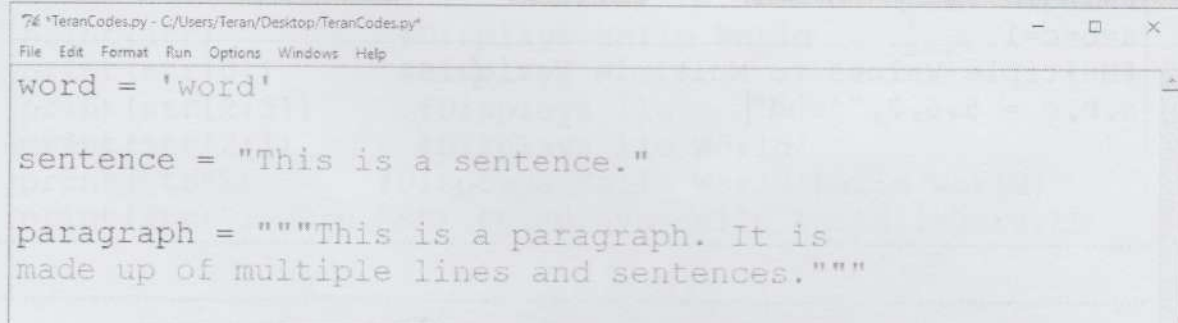


```
if True:
    print ("ITGuru.lk")
    print ("Good Morning")
else:
    print ("Teran.lk")
    print ("Good Afternoon")
```

### Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal –



```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

### Comments in Python

**Single Line Comment** - A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

**Multi Line Comments** – Require to have multiple # signs on each line.

## Variables

- Variables are nothing but reserved memory locations to store values.
- This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the 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.
- Python variables do not need explicit declaration to reserve memory space.

## Assigning Values to Variables

- The equal sign (=) is used to assign values to variables.

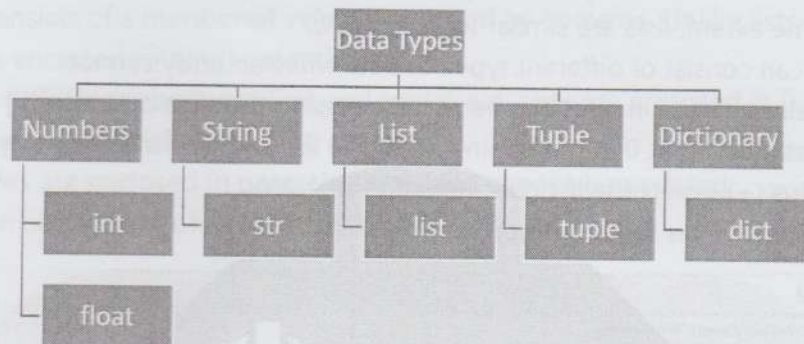
Works from..

## Multiple Assignments

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

```
TeranCodes.py - C:/Users/Terani/Desktop/TeranCodes.py
File Edit Format Run Options Windows Help
#Single Value to Several Variables EX:-
a=b=c=1
#Multiple Values to Multiple Variables
a,b,c = 5.6,2,"john"
```

## Data Types



### Strings

- Python allows for either pairs of single or double quotes to define a string
- Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator
- The asterisk (\*) is the repetition operator.

```
74 *TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py
File Edit Format Run Options Windows Help
str = 'Hello World!'

print(str)           #Displays Hello World
print(str[0])        #Displays H
print(str[2:5])      #Displays llo
print(str[2:])       #Displays llo World!
print(str*2)         #Displays Hello World!Hello World!
print(str+"itguru.LK") #Displays Hello World!itguru.LK
```



## Lists

- A list contains items separated by commas and enclosed within square brackets (`[]`).
- To some extent, lists are similar to arrays in C.
- A List can consist of different types of data while an array cannot.
- The values stored in a list can be accessed using the slice operator (`[]` and `[:]`) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator
- The asterisk (\*) is the repetition operator.

## Source Code

```
74 TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py
File Edit Format Run Options Windows Help
mylist = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(mylist)          # Prints complete list
print(mylist[0])        # Prints first element of the list
print(mylist[1:3])      # Prints elements starting from 2nd till 4th
print(mylist[2:])       # Prints elements starting from 3rd element
print(tinylist * 2)     # Prints list two times
print(mylist + tinylist) # Prints concatenated lists
mylist[0]="angi"        # Changes the first element of the list
print(mylist[0])        # Prints first element of the list again
```

## Output

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
angi
```

## Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within () parentheses.
- The main differences between lists and tuples are: Lists are enclosed in [] brackets and their elements and size can be changed.
- while tuples are enclosed in parentheses () and cannot be updated.
- Tuples can be thought of as read-only lists.

## Source Code

```
File Edit Format Run Options Windows Help
mytuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print(mytuple)          # Prints complete list
print(mytuple[0])       # Prints first element of the list
print(mytuple[1:3])     # Prints elements starting from 2nd till 3rd
print(mytuple[2:])      # Prints elements starting from 3rd element
print(tinytuple * 2)    # Prints list two times
print(mytuple + tinytuple) # Prints concatenated lists
mytuple[0]="angi"       #Raise an Error
```

## Output

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
Traceback (most recent call last):
  File "C:/Users/Teran/Desktop/TeranCodes.py", line 10, in <mod
ule>
    mytuple[0]="angi"          #Raise an Error
TypeError: 'tuple' object does not support item assignment
```

## Dictionary

- Work like associative arrays or hashes found in Perl and consist of key-value pairs.
- A dictionary key can be almost any Python type but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces {} and values can be assigned and accessed using square braces [].

## Source Code

```
TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py
File Edit Format Run Options Windows Help

mydict = {} #Creates an Empty Dictionary
mydict['one'] = "This is one" #Key and the Value
mydict[2] = "This is two" #Key and the Value

tinydict = {'name': 'john', 'code': 6734, 'dept': 'sales'}

print(mydict['one'])      # Prints value for 'one' key
print(mydict[2])          # Prints value for 2 key
print(tinydict)           # Prints complete dictionary
print(tinydict.keys())    # Prints all the keys
print(tinydict.values())  # Prints all the values
```

## Output

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
dict_keys(['dept', 'code', 'name'])
dict_values(['sales', 6734, 'john'])
```



### Type Conversions

To perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

- `int(x)`
- `float(x)`
- `str(x)`

Take out your writing Books.....

### Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression  $4 + 5 = 9$ . Here, 4 and 5 are called operands and + is called operator.

#### Types of Operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

#### Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then -

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 30$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -10$
* Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/ Division	Divides left hand operand by right hand operand	$b / a = 2$

% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$
** Exponent	Performs exponential (power) calculation on operators	$a^{**}b = 10 \text{ to the power } 20$
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) -	$9//2 = 4$ and $9.0//2.0 = 4.0$ $-11//3 = -4$ $-11.0//3 = -4.0$

### Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then -

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	$(a == b)$ is not true
!=	If values of two operands are not equal, then condition becomes true.	$(a != b)$ is true.
<>	If values of two operands are not equal, then condition becomes true.	$(a <> b)$ is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	$(a > b)$ is not true
<	If the value of left operand is less than the value of right operand, then condition becomes true.	$(a < b)$ is true.

<b>&gt;=</b>	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<b>&lt;=</b>	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

### Assignment Operators

Assume variable a holds 10 and variable b holds 20, then –

Operator	Description	Example
<b>=</b>	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
<b>+= Add AND</b>	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
<b>-= Subtract AND</b>	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
<b>*= Multiply AND</b>	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
<b>/= Divide AND</b>	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a c /= a is equivalent to c = c / a
<b>%= Modulus AND</b>	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
<b>**= Exponent AND</b>	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
<b>//= Floor Division</b>	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a



### Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if  $a = 60$ ; and  $b = 13$ ; Now in binary format they will be as follows –

$a = 0011\ 1100$

$b = 0000\ 1101$

-----  
 $a \& b = 0000\ 1100$

$a | b = 0011\ 1101$

$a \wedge b = 0011\ 0001$

$\sim a = 1100\ 0011$

Operator	Description	Example
$\&$ Binary AND	Operator copies a bit to the result if it exists in both operands	$(a \& b)$ (means $0000\ 1100$ )
$ $ Binary OR	It copies a bit if it exists in either operand.	$(a   b) = 61$ (means $0011\ 1101$ )
$\wedge$ Binary XOR	It copies the bit if it is set in one operand but not both.	$(a \wedge b) = 49$ (means $0011\ 0001$ )
$\sim$ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	$(\sim a) = -61$ (means $1100\ 0011$ in 2's complement form due to a signed binary number.
$<<$ Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	$a << 2 = 240$ (means $1111\ 0000$ )
$>>$ Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	$a >> 2 = 15$ (means $0000\ 1111$ )

## Logical Operators

There are following logical operators supported by Python language.

Assume variable a holds 10 and variable b holds 20 then –

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	not(a and b) is false.

## Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

## Python Operators Precedence

.....

.....

.....

.....

## Decision Making

Python programming language assumes any non-zero and non-null values as TRUE, and if it is either zero or null, then it is assumed as FALSE value.

- There can be

### Single Statements

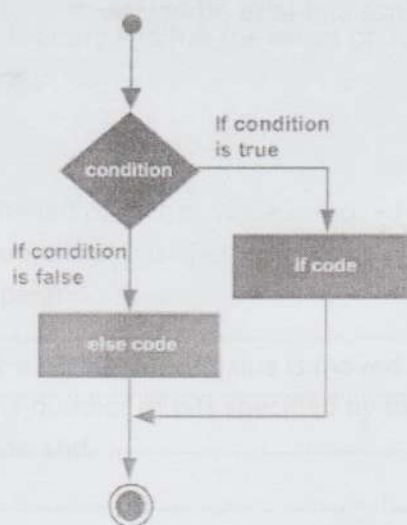
#### Source Code

```
74 TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py
File Edit Format Run Options Windows Help
var = 100
if (var == 100) : print("Value of expression is 100")
else: print("Not 100")
print("Good bye!")
```

#### Output

```
Value of expression is 100
Good bye!
```

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.





## else if OR elif

- The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the **else**, the **elif** statement is optional.

### Source Code - elif

```
"I\TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py"
File Edit Format Run Options Windows Help
mark=55
if (mark>70):
    print("ITGURU.LK - Distinction")
elif (mark>50):
    print("ITGURU.LK - Credit")
elif (mark>30):
    print("ITGURU.LK - Simple")
else:
    print("Fail")
print("Bye")
```

### OR USING else if

```
"I\TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py"
File Edit Format Run Options Windows Help
mark=55
if (mark>70):
    print("ITGURU.LK - Distinction")
else:
    if (mark>50):
        print("ITGURU.LK - Credit")
    else:
        if (mark>30):
            print("ITGURU.LK - Simple")
        else:
            print("Fail")
print("Bye")
```

### Output

```
ITGURU.LK - Credit
Bye
```

---

---

## Loops – Iteration

- executes a block of code several number of times
- .....
- Python uses for loop and while loop

### While Loop and For Loop

#### Source Code – While Loop

```
74 "TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py"
File Edit Format Run Options Windows Help
count = 0
while (count < 5):
    print('The count is:', count)
    count = count + 1

print("Good bye!")
```

#### Source Code – For Loop – Method 1

```
74 "TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py"
File Edit Format Run Options Windows Help
for count in range(5):
    print('The count is:', count)
    count = count + 1

print("Good bye!")
```

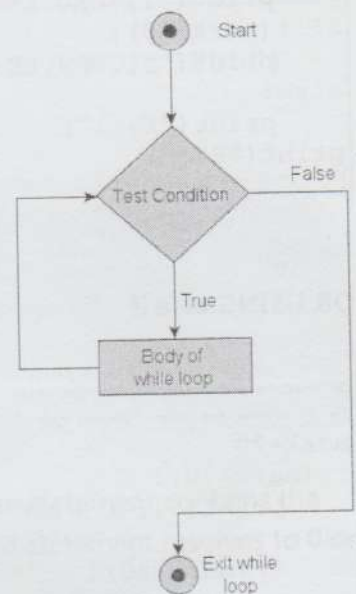
#### Source Code – For Loop – Method 2

```
74 "TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py"
File Edit Format Run Options Windows Help
for count in range(0,5):
    print('The count is:', count)
    count = count + 1

print("Good bye!")
```

#### Output – Common

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
Good bye!
```



## Intelligent for loop

### Source Code

74 \*TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py\*

File Edit Format Run Options Windows Help

```
mystring="itguru.lk"
```

```
mylist=["John","Angi","Hafsa","Paul"]
```

```
for i in mystring:  
    print(i)
```

```
print("-----")
```

```
for name in mylist:  
    print(name)
```

### Output

```
i  
t  
g  
u  
r  
u  
.  
l  
k
```

```
-----  
John  
Angi  
Hafsa  
Paul
```





## Loop Control Statements

### Source Code - Break

```
74 *TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py*  
File Edit Format Run Options Windows Help  
for i in range(1,10):  
    print(i)  
    if(i==3):  
        break  
    print("bye")
```

### Output

```
1  
bye  
2  
bye  
3
```

### Source Code - Continue

```
74 *TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py*  
File Edit Format Run Options Windows Help  
for i in range(1,5):  
    print(i)  
    if(i==3):  
        continue  
    print("bye")
```

### Output

```
1  
bye  
2  
bye  
3  
4  
bye
```

### Source Code - Pass

```
74 *TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py*  
File Edit Format Run Options Windows Help  
for i in range(1,5):  
    print(i)  
    if(i==3):  
        pass  
    print("bye")
```

### Output

```
1  
bye  
2  
bye  
3  
bye  
4  
bye
```

## Sub Programs – Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Python has Built-in functions, in addition we can define functions also which will be known as user-defined functions.

### Source Code

```
74 *TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py*
File Edit Format Run Options Windows Help
#function without return
def displayName(first,last):
    print(first+last)

#function with return
def giveFullName(first,last):
    return first+last

#Default Value for num3 is 10
def average(num1,num2,num3=10):
    avg=(num1+num2+num3)/3
    return avg

#calling a function
displayName("John","Silva")
print(giveFullName("Angi","Fernando"))
fullName=giveFullName("Teran","Subasinghe")
print(fullName)
print(average(8,6,4))
print(average(8,6))
```

### Output

```
JohnSilva
AngiFernando
TeranSubasinghe
6.0
8.0
```



## File Handling

- The `open()` function can be used to open a file object in Python.
- The `open()` function takes two parameters; filename, and mode.
- There are four different modes:
  - "r" - Read - Default value. Opens a file for reading, error if the file does not exist
  - "a" - Append - Opens a file for appending, creates the file if it does not exist
  - "w" - Write - Opens a file for writing, creates the file if it does not exist

### Source Code

```
74 *TeranCodes.py - C:/Users/Teran/Desktop/TeranCodes.py*
File Edit Format Run Options Windows Help
fw1=open("test.txt","w")
fw1.write("John\n")
fw1.write("Angi\n")
fw1.write("Paul\n")
fw1.close()

fa=open("test.txt","a")
fa.write("Hafsa\n")
fa.write("Gopi\n")
fa.close()

fr=open("test.txt","r")
for i in fr:
    print(i)
fr.close()

#Replacing the test.txt
fw2=open("test.txt","w")
fw2.close()
```

### Output

John

Angi

Paul

Hafsa

Gopi

### Text File - Trace