PYTHON-BASIC

BY AMAR PANCHAL

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

PYTHON IS

- **Python is a high-level, interpreted, interactive and object-oriented scripting language. Python was designed to be highly readable which uses English keywords frequently whereas other languages use punctuation and it has fewer syntactical constructions than other languages. Created by Guido van Rossum and first released in 1991**

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

History of Python

- December 1989 I Implementation by Guido van Rossum as successor of ABC to provide exception handling
- February 1991 I Python 0.9.0 had classes with inheritance, exception handling, functions, and the core datatypes
- January 1994 I Version 1.0 has functional programming features
- October 2000 I Python 2.0 brings garbage collections
- Python 2.2 improves Python's types to be purely object oriented
- December 2008 I Python 3 I Reduce feature duplication by removing old ways of doing things.
- **Not backwards compatible with Python 2.x**

History of Python

- Modules
  - Reusable pieces of software
  - Can be written by any Python developer
  - Extend Python's capabilities
- Python Web site at www.python.org
  - Primary distribution center for Python source code, modules and documentation

PYTHON IS GREAT BECAUSE...

- **Portable**
  - Your program will run if the interpreter works and the modules are installed
- **Efficient**
  - Rich language features built in
  - Simple syntax for ease of reading
  - Focus shifts to "algorithm"
- **Flexible**
  - Imperative
  - Object-oriented
  - Functional

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

PYTHON IS GREAT BECAUSE...

- **Extendible**
  - Plenty of easy-to-use modules
  - If you can imagine it, then someone has probably developed a module for it
  - Your program can adjust the way the interpreter works
- **Python syntax**
  - Extremely simplified syntax
  - Nearly devoid of special characters
  - Intended to be nearly English
- **Dynamic types**
  - It is the responsibility of the programmer
  - Still "strongly typed

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

INDUSTRIAL USE OF PYTHON

- **Web Development**
- **Games**
- **Graphics**
- **Financial**
- **Science**
- **Electronic Design Automation**
- **Education**
- **Business Software**

VERSION PROBLEM

PYTHON 2.X	PYTHON 3.X
started in 2000 and slated to lose all support in 2020	Started in 2008 and seen as future of Python
Absence of new modules	Newer modules like __future__ and asy nio
print keyword	print() is a function
ASCII and UNICODE support Use u for Unicode u 'Hello‘ else byte sequence	UNICODE default All strings are now Unicode b used for byte sequence
Division With Integers 5 / 2=2	Division With Integers 5 / 2=2.5
raw_input() function and an input()	Only input()

INTERACTIVE MODE

- "copyright“
- "credits“
- "license()”

Running

- We assume that you have the Python interpreter set in PATH variable. Now, try to run this program as follows-
- On Linux: $ python test.py

This produces the following result-

- Hello, Python!
- On Windows
- C:\Python34>Python test.py

This produces the following result-

- Hello, Python!

Python Identifiers

- 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 does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language

Naming conventions for python identifiers

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strong private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language defined special name.

Key Words

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

Prof.Amar P

Lines and Indentation

- Python does not use braces({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by l
- if True:
- print ("True")
- else:
- print ("False") ine indentation, which is rigidly enforced.

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Multi-Line Statements

- Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (\) to denote that the line should continue.

```
total = item_one + \
        item_two + \
        item_three
```

The statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example-

```
days = ['Monday', 'Tuesday', 'Wednesday',
            'Thursday', 'Friday']
```


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.

Comments in Python

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

- # First comment
- print ("Hello, Python!") # second comment

Waiting for the User

- input("\Press the enter key to exit.")

Multiple Statements on a Single Line

- The semicolon (;) allows multiple statements on a single line given that no statement starts a new code block.
- Example

Suites

- Groups of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite
- Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite.

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

Your First Python Program

- At the prompt (>>>) type:

```
print "Game Over"
```

  - Press [Enter]

- Very straightforward
  - You could have guessed what this does without knowing Python!

Your First Python Program

- Python is “ ”
- TRY
  - print “Game Over”
  - Print “Game Over”
  - PRINT “Game Over”

PYTHON-VARIABLES

BY AMAR PANCHAL

Assigning Values to Variables

- Python variables do not need explicit declaration 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.
- Python allows you to assign a single value to several variables simultaneously.
- EXAMPLE
- Special case also
  - a, b, c = 24, 36, “AMAR”

Standard Data Types

- Python has five standard data types-
  - Numbers
  - Boolean
  - String
  - List
  - Tuple
  - Dictionary
  - Set

Numbers

- int (signed integers)
- float (floating point real values)
- complex (complex numbers)

Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes. 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 to the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator
- str = 'This is a String'

- print (str)
- print (str[0])
- print (str[2:5])
- print (str[2:])
- print (str * 2)
- print (str + "TEST")

Lists

- A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.
- 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, and the asterisk (*) is the repetition operator.

```
list = [ 'amp', 4624 , 3.14, 'python', 0.07 ]
tinylist = [420, 'abcd']
print (list)
print (list[0])
print (list[1:3])
print (list[2:])
print (tinylist * 2)
print (list + tinylist)
```

28

LIST

Function	Description
all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list
sorted()	Return a new sorted list (does not sort the list itself).
sum()	Return the sum of all elements in the list.

29

- fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']

```
>>> fruits.count('apple')
>>> fruits.count('tangerine')
>>> fruits.index('banana')
>>> fruits.index('banana', 4)
>>> fruits.reverse()
>>> fruits
>>> fruits.append('grape')
>>> fruits
```

30

Tuples

- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parenthesis.
- The main difference 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.

```
tuples = ( ‘amp', 4624 , 3.14, ‘python', 0.07 )
tinylist = (420, ‘abcd’)
print (list)
print (list[0])
print (list[1:3])
print (list[2:])
print (tinylist * 2)
print (list + tinylist)
```

31


```
>>>my_tuple = ('p','e','r','m','i','t')

>>>print(my_tuple[0])

>>>print(my_tuple[5])

# nested tuple
>>>n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

>>>print(n_tuple[0][3])



>>>print(n_tuple[1][1])
```

32

Negative Indexing

```
>>>my_tuple = ('p','e','r','m','i','t')
>>>print(my_tuple[-1])
```

Slicing

We can access a range of items in a tuple by using the slicing operator - colon ":".

```
>>>my_tuple = ('p','r','o','g','r','a','m','i','z')

>>>print(my_tuple[1:4])


>>>print(my_tuple[:-7])


>>>print(my_tuple[7:])


>>>print(my_tuple[:])


>>>my_tuple = (4, 2, 3, [6, 5])
```

33


```
# we cannot change an element
# you will get an error:
# TypeError: 'tuple' object does not support item assignment

>>>my_tuple[1] = 9

# but item of mutable element can be changed
>>>my_tuple[3][0] = 9


# tuples can be reassigned
>>>my_tuple = ('p','r','o','g','r','a','m','i','z')
>>>print(my_tuple)
```

5/1/2019

34

del

The del statement

CODE:

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a

>>> del a[2:4]
>>> a

>>> del a[:]
>>> a

>>> del a
>>>print(a)
```

35

Dictionary

- Python's dictionaries are kind of hash-table type. They 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 ([]).
- dict = {}
- dict['one'] = "This is one"
- dict[2] = "This is two"
- tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
- print (dict['one'])
- print (dict[2])
- print (tinydict)
- print (tinydict.keys())
- print (tinydict.values())

36

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v(defaults to None).
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

37

Set

- A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

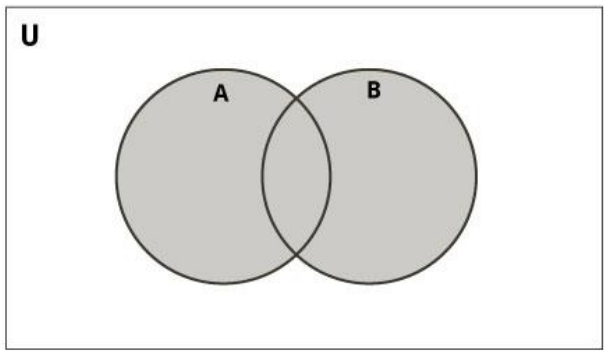
38

Python Set Methods	
Method	Description
add()	Add an element to a set
clear()	Remove all elements form a set
copy()	Return a shallow copy of a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set
discard()	Remove an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return True if two sets have a null intersection
issubset()	Return True if another set contains this set
issuperset()	Return True if this set contains another set
pop()	Remove and return an arbitary set element. Raise KeyErrorif the set is empty
remove()	Remove an element from a set. If the element is not a member, raise a KeyError
symmetric_difference()	Return the symmetric difference of two sets as a new set
symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

39

Set Union

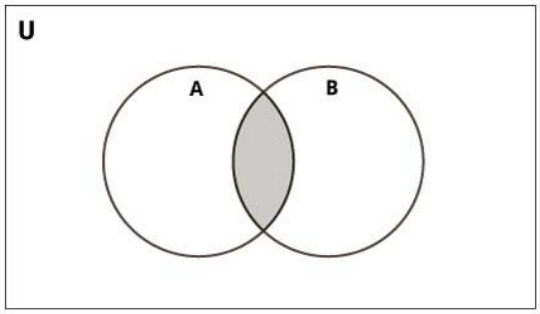
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use | operator
print(A | B)
```



40

Set Intersection

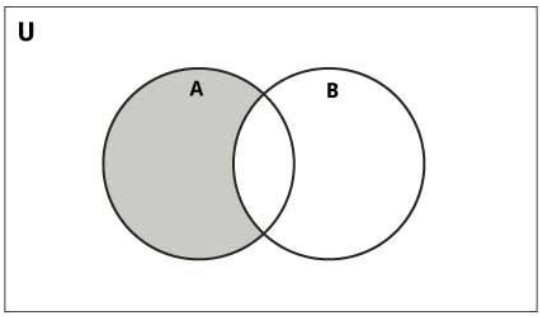
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
# use & operator#
print(A & B)
```



41

Set Difference

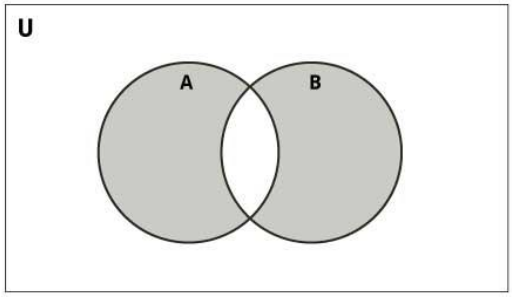
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A - B)
```



42

Set Symmetric Difference

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A ^ B)
```



43

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0.7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

Data Type Conversion

Function	Description
int(x [,base])	Converts x to an integer. The base specifies the base if x is as string.
float(x)	Converts x to a floating-point number.
complex(real[,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string
oct(x)	Converts an integer to an octal string.

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

PYTHON-OPERATORS

BY

AMAR PANCHAL

Types of Operator

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

Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 31
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 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

CODE

```
a = 21
b = 10
c = 0

c = a + b
print ("Line 1 - Value of c is ", c)
c = a - b
print ("Line 2 - Value of c is ", c )
c = a * b
print ("Line 3 - Value of c is ", c)
c = a / b
print ("Line 4 - Value of c is ", c )
c = a % b
print ("Line 5 - Value of c is ", c)

a = 2
b = 3
c = a**b
print ("Line 6 - Value of c is ", c)

a = 10
b = 5
c = a//b
print ("Line 7 - Value of c is ", c)
```

Comparison Operators

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 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.
>=	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.
<=	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.

CODE

```
a = 21
b = 10

if ( a == b ):
   print ("Line 1 - a is equal to b")
else:
   print ("Line 1 - a is not equal to b")

if ( a != b ):
   print ("Line 2 - a is not equal to b")
else:
   print ("Line 2 - a is equal to b")

if ( a < b ):
   print ("Line 3 - a is less than b" )
else:
   print ("Line 3 - a is not less than b")
```

```
if ( a > b ):
   print ("Line 4 - a is greater than b")
else:
   print ("Line 4 - a is not greater than b")

a,b=b,a #values of a and b swapped. a becomes 10, b becomes 21

if ( a <= b ):
   print ("Line 5 - a is either less than or equal to  b")
else:
   print ("Line 5 - a is neither less than nor equal to  b")

if ( b >= a ):
   print ("Line 6 - b is either greater than  or equal to b")
else:
   print ("Line 6 - b is neither greater than  nor equal to b")
```


Assignment Operators

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

```
a = 21
b = 10
c = 0

c = a + b
print ("Line 1 - Value of c is ", c)

c += a
print ("Line 2 - Value of c is ", c )

c *= a
print ("Line 3 - Value of c is ", c )

c /= a
print ("Line 4 - Value of c is ", c )

c  = 2
c %= a
print ("Line 5 - Value of c is ", c)

c **= a
print ("Line 6 - Value of c is ", c)

c //= a
print ("Line 7 - Value of c is ", c)
```


Bitwise Operators

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)
^ Binary XOR	It copies the bit, if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operand's value is moved left by the number of bits specified by the right operand.	a << = 240 (means 1111 0000)
>> Binary Right Shift	The left operand's value is moved right by the number of bits specified by the right operand.	a >> = 15 (means 0000 1111)

```
a = 60
b = 13

print ('a=',a,':',bin(a),'b=',b,':',bin(b))
c = 0

c = a & b;
print ("result of AND is ", c,':',bin(c))

c = a | b;
print ("result of OR is ", c,':',bin(c))

c = ~a;
print ("result of COMPLEMENT is ", c,':',bin(c))

c = a << 2;
print ("result of LEFT SHIFT is ", c,':',bin(c))

c = a >> 2;
print ("result of RIGHT SHIFT is ", c,':',bin(c))
```


Logical Operators

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is False.
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 True.

Membership Operators

- Python's membership operators test for membership in a sequence, such as strings, lists, or tuples.

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.


```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ]

if ( a in list ):
   print ("Line 1 - a is available in the given list")
else:
   print ("Line 1 - a is not available in the given list")

if ( b not in list ):
   print ("Line 2 - b is not available in the given list")
else:
   print ("Line 2 - b is available in the given list")
```

```
c=b/a
if ( c in list ):
   print ("Line 3 - a is available in the given list")
else:
   print ("Line 3 - a is not available in the given list")
```


Identity Operators

- Identity operators compare the memory locations of two objects

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not**results in 1 if id(x) is not equal to id(y).

```
a = 20
b = 20
print ('Line 1','a=',a,':',id(a), 'b=',b,':',id(b))

if ( a is b ):
   print ("Line 2 - a and b have same identity")
else:
   print ("Line 2 - a and b do not have same identity")

if ( id(a) == id(b) ):
   print ("Line 3 - a and b have same identity")
else:
   print ("Line 3 - a and b do not have same identity")
```



```
b = 30
print ('Line 4','a=',a,':',id(a), 'b=',b,':',id(b))

if ( a is not b ):
   print ("Line 5 - a and b do not have same identity")
else:
   print ("Line 5 - a and b have same identity")
```

Operators Precedence

S.No.	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % // Multiply, divide, modulo and floor division
4	+ - Addition and subtraction
5	>> << Right and left bitwise shift
6	& Bitwise 'AND'

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Operators Precedence

S.No.	Operator & Description
7	^ | Bitwise exclusive `OR' and regular `OR'
8	<= < > >= Comparison operators
9	<> == != Equality operators
10	= %= /= //= -= += *= **= Assignment operators
11	is is not Identity operators
12	in not in Membership operators
13	not or and Logical operators

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

PYTHON

CONDITIONAL STSTEMENT & LOOPS

BY

AMAR PANCHAL

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

IF

if Statements	It consists of a Boolean expression which results is either TRUE or FALSE followed by one or more statements.

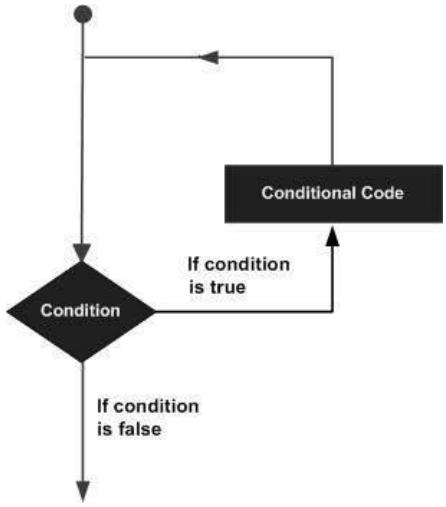
```
if expression:
        #execute your code

if expression:
        #execute your code
else:
    #execute your code
```

```
if expression:
        #execute your code
elif expression:
        #execute your code
else:
        #execute your code
```

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Loops

A loop statement allows us to execute a statement or group of statements multiple times.
The following diagram illustrates a loop statement –



While

- Syntax

```
while expression:
        statement(s)
```

Ex

```
count = 0
while (count < 5):
   print ('The count is:', count)
   count = count + 1

print ("Good bye!")
```

for Loop

- Syntax

for iterating_var in sequence:

  statements(s)

- The range() function

The built-in function range() is the right function to iterate over a sequence of numbers. It generates an iterator of arithmetic progressions.

Example

>>> range(5)

For Loop

- For i in range(5):

    print(i)

- For i in “python”:

    print(i)

Loop Control Statements

S.No.	Control Statement & Description
1	**break statement:** Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	**continue statement:** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	**pass statement:** The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.


```
for letter in 'Python':
    if letter == 'h':
        break
    print ('Current Letter :', letter)
```

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

```
var = 5
while var > 0:
   print ('Current variable value :', var)
   var = var -1
   if var == 2:
         break

print ("Good bye!")
```



```
for letter in 'Python':
   if letter == 'h':
      pass
      print ('This is pass block')
   print ('Current Letter :', letter)

print ("Good bye!")
```

Single Statement Suites

```
flag = 1
while (flag): print ('Given flag is really true!')
print ("Good bye!")
```


else block with loops

```
no = int(input('any number: '))
numbers = [11,33,55,39,55,75,37,21,23,41,13]
    for num in numbers:
      if num == no:
            print ('number found in list')
            break
    else:
      print ('number not found in list')
```

```
numbers = [11,33,55,39,55,75,37,21,23,41,13]

for num in numbers:
   if num%2 == 0:
      print ('the list contains an even number')
      break
else:
   print ('the list does not contain even number')
```


Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Python-Functions

BY

AMAR PANCHAL

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

FUNCTIONS

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Defining a Function

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( ))

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```


Pass by Reference vs Value

Function Arguments

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

The Anonymous Functions

- You can use the **lambda**keyword to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression.
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Syntax
- lambda [arg1 [,arg2,.....argn]]:expression

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Scope of Variables

- Global variables
- Local variables


```
total = 0

def sum( arg1, arg2 ):
   total = arg1 + arg2; # Here total is local variable.
   print ("Inside the function local total : ", total)
   return total

sum( 10, 20 )
print ("Outside the function global total : ", total )
```

PYTHON –OOPS

BY AMAR PANCHAL

Creating Classes

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

Creating Instance Objects

Accessing Attributes

- The getattr(obj, name[, default]) – to access the attribute of object.
- The hasattr(obj,name) – to check if an attribute exists or not.
- The setattr(obj,name,value) – to set an attribute. If attribute does not exist, then it would be created.
- The delattr(obj, name) – to delete an attribute.

Importing

Built-In Class Attributes

- **__dict__** – Dictionary containing the class's namespace.
- **__doc__** – Class documentation string or none, if undefined.
- **__name__** – Class name.
- **__module__** – Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- **__bases__** – A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Class Inheritance

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
  'Optional class documentation string'
  class_suite
```


Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

check a relationships

- The **issubclass(sub, sup)** boolean function returns True, if the given subclass **sub** is indeed a subclass of the superclass **sup.**

- The **isinstance(obj, Class)** boolean function returns True, if *obj* is an instance of class *Class* or is an instance of a subclass of Class

Overriding Methods

S.No.	Method, Description & Sample Call
1	__init__ (self [,args...]) Constructor (with any optional arguments) Sample Call : obj = className(args)
2	__del__(self) Destructor, deletes an object Sample Call : del obj
3	__repr__(self) Evaluatable string representation Sample Call : repr(obj)
4	__str__(self) Printable string representation Sample Call : str(obj)
5	__cmp__ (self, x) Object comparison Sample Call : cmp(obj, x)

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

Data Hiding

- The __ sign

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

PYTHON-EXCEPTIONS

BY

AMAR PANCHAL

EXCEPTION HANDLING

- Exception & Assertions

Handling an exception

```
try:
   You do your operations here
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```


The except Clause with Multiple Exceptions

```
try:
   You do your operations here
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block
```

The try-finally Clause

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be skipped.
finally:
   This would always be executed.
   ......................
```


Argument of an Exception

```
try:
   You do your operations here
   ......................
except ExceptionType as Argument:
   You can print value of Argument here...
```

Raising an Exception

- Syntax

raise [Exception [, args [, traceback]]]

User-Defined Exceptions

```
class MyError(Exception):
      def __init__(self, value):
                  self.value = value
      def __str__(self):
                  return self.value
try:
       raise MyError(2*2)
       except MyError, e:
       print("My exception occurred:", e.value)
```

Assertions in Python

- The syntax for assert is –

  assert Expression[, Arguments]

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

PYTHON-FILES

BY

AMAR PANCHAL

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

The open Function

- file object = open(file_name [, access_mode][, buffering])

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com

modes

MODE	
r	
r+	
w	
w+	
a	
a+	
rb	
wb	
ab	
rb+	
wb+	
ab+	

The file Object Attributes

S.No.	Attribute & Description
1	**file.closed** Returns true if file is closed, false otherwise.
2	**file.mode** Returns access mode with which file was opened.
3	**file.name** Returns name of the file.

code

```
f = open(“MyFile", “w")
print ("Name of the file: ", f.name)
print ("Closed or not : ", f.closed)
print ("Opening mode : ", f.mode)
f.close()
```


The close()

- fileObject.close()

Reading and Writing Files

- The **write()** method writes any string to an open file.

```
fileObject.write(string);
f = open("MyFile.txt", "w")
f.write( "file operation writing process")
f.close()
```


Reading and Writing Files

- The **read()** method reads a string from an open file.
- fileObject.read([count]);

File Positions

tell()	
seek(offset[, from])	

Renaming and Deleting Files

- Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.To use this module, you need to import it.

rename()	**os.rename(current_file_name, new_file_name)**
remove()	os.remove(file_name)

Dealing With Directories

Command	Use
os.mkdir("newdir")	
os.chdir("newdir")	
os.getcwd()	
os.rmdir(“dirname”)	

Prof.Amar Panchal | 9821601163 | www.amarpanchal.com