PYTHON : QUICK REVISION TOUR

Introduction to Python

- ✓ Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable.
- ✓ **Designed by** : **Guido Van Rossum** in **1990**
- ✓ **Developer: Python Software Foundation**
- ✓ **Website (to download)** : **http://www.python.org**

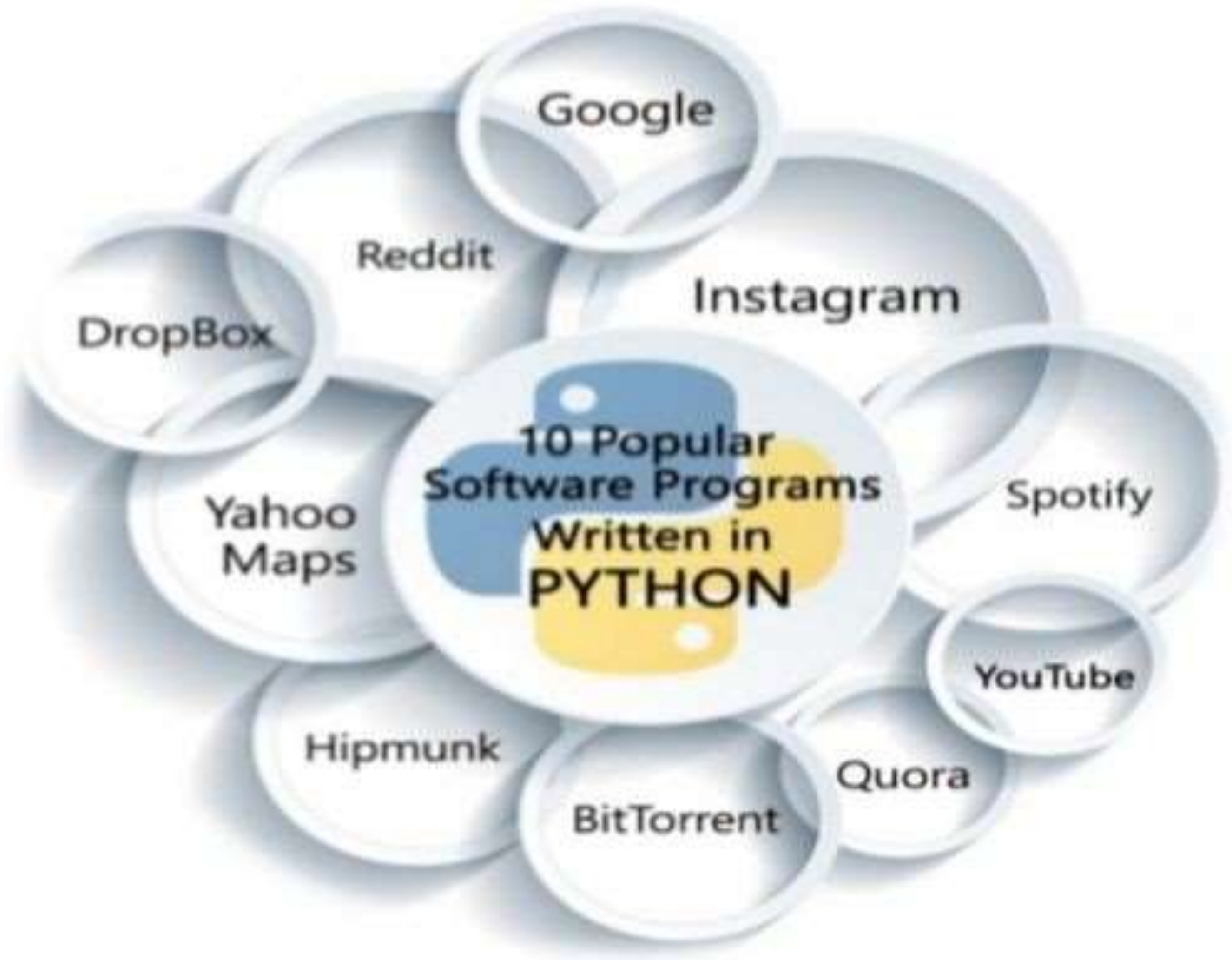
Features of Python

- ✓ **Portablity** –Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- ✓ **Extendablity** – It allows to add low-level modules to the Python interpreter.
- ✓ **Databases** –Python provides interfaces to all major commercial databases.
- ✓ **GUI Programming** –Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the XWindow system of Unix.
- ✓ **Very Easy-to-learn** –Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- ✓ **Readability** –Python code is more clearly defined and visible to the eyes.
- ✓ **Interactive Mode** –Python uses SHELL which allows interactive testing and debugging of snippets of code.
- ✓ **Garbage Collections** :It supports automatic garbage collection.
- ✓ It can be **easily integrated** with C, C++, COM, ActiveX, CORBA, and Java.

POPULAR SOFTWARES MADE IN PYTHON

- DROP BOX
- GOOGLE
- INSTAGRAM
- YOUTUBE
- BIGTORRENT
- YAHOO MAPS

etc



Python Character Set

- Character Set means characters and symbols which can be determined by python language.
- A character set consist of characters, symbols, digits, or any other symbolic character.
  - Letters: A–Z, a–z
  - Digits: 0–9
  - Special Symbols: _, +, −, *, /, (,), [,] . . . Etc.
  - White Spaces: blank space, tab, carriage return, newline, form feed etc.
  - Other characters: python can process all kind of ASCII and UNICODE characters.

Tokens

- Smallest units of any programming language are called Tokens (Lexical Unit) Python tokens are–
  i. Keywords
  ii. Identifiers (Names)
  iii. Literals
  iv. Operators
  v. Punctuators

Keywords

- Keywords are reserve words for compiler and has special meaning for compiler.

Keywords in Python				
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Identifiers

- A **Python identifier** is a name used to identify a variable, function, class, module or other object

  - Python identifier may be combination of alphabets, underscore and digits.
  - The first letter of an identifier must be a letter or underscore (_) .
  - Case sensitive (Upper case letter and lower case letter are different.)
  - Length of Identifier is endless and case sensitive.
  - No keyword can be taken as identifier.

Literals

- Literals are often called Constant Values.
- Python allows different types of literals –
  - String literals – “Manasvi ”,’ F’ ,”24”
  - Numeric literals – 10, 13.5, 3+5i
  - Boolean literals – True or False
  - Special Literal ***None***

String Literals

- String Literal is a sequence (collection) of characters enclosed with Single (‘ ‘) , double(“ “) or triple quote(‘’ ’ ‘’ ’)
- A=“Hello”
- Name=“”” Tani”””
- X= ‘Gwalior’

Numeric Literals

- Numeric Literals can be of 3 types–
  - int (signed integers)
    - Decimal Integer Literals – 10, 17, 210
    - Octal Integer Literals – 0o17, 0o217 etc.
    - Hexadecimal Integer Literals – 0x14, 0x2A4, 0xABD etc.
  - float (floating point real value)
    - Fractional Form – 2.0, 17.5 –13.5, –.00015 etc.
    - Exponent Form – –1.7E+8, .25E–4 etc.
  - complex (complex numbers)
    - 3+5i etc.

Boolean Literals

- True
- False

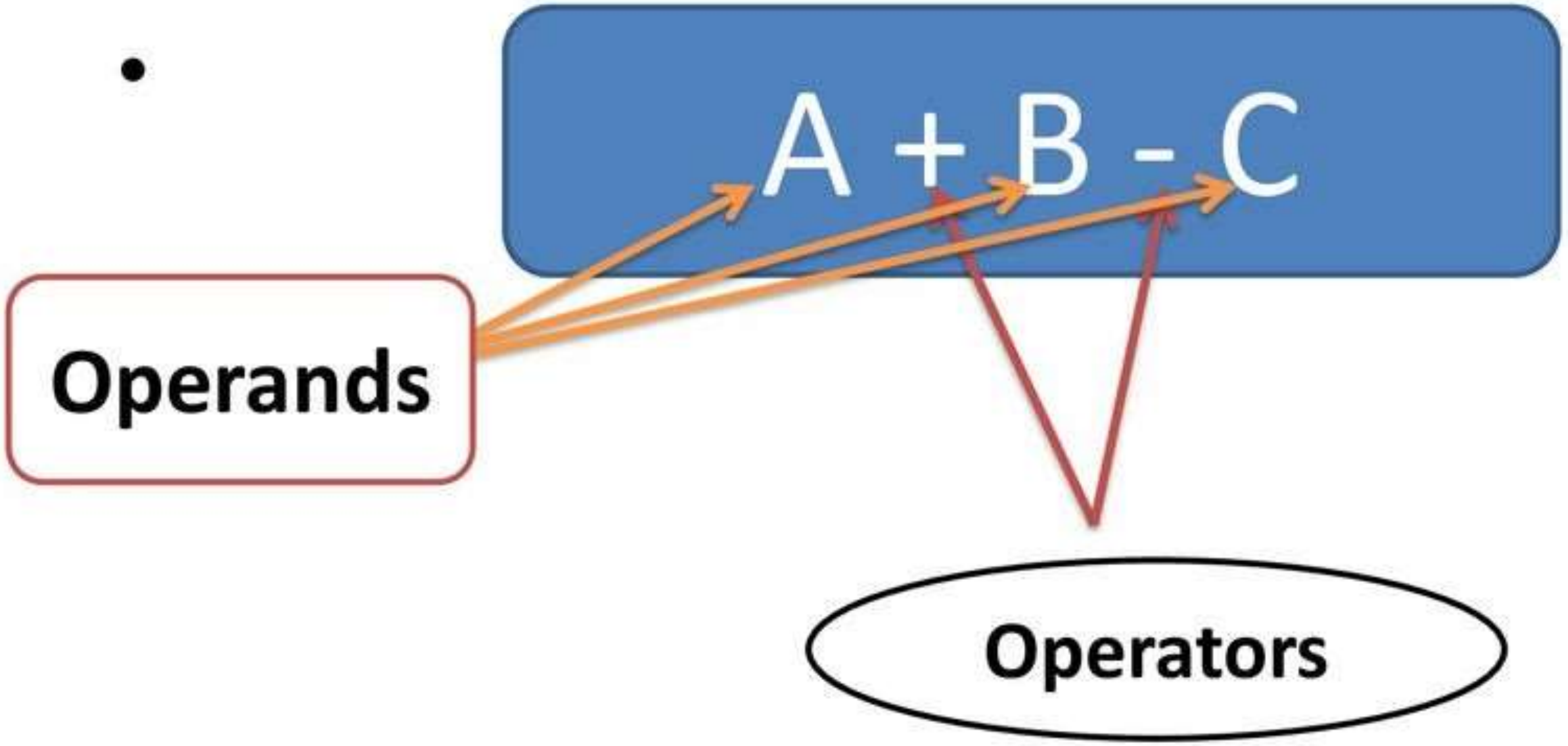
Special Literals

- None

(means nothing)

OPERATORS

- Operators are symbols used to perform operations on values and variables.
-



Types of Operators

- Unary Operator
  - Unary plus (+)
  - Unary Minus (−)
  - Bitwise complement (~)
  - Logical Negation (not)
- Binary Operator
  - Arithmetic operator (+, −, *, /, %, **, //)
  - Relational Operator(<, >, <=, >=, ==, !=)
  - Logical Operator (and, or)
  - Assigment Operator (=, /=, +=, −=, *=, %=, **=, //=)
  - Bitwise Operator (& *bitwise and*, ^ *bitwise xor*, | *bitwise or*)
  - Shift operator (<< *shift left*, >> *shift right*)
  - Identity Operator (is, is not)
  - Membership Operator (in, not in)

Lets Practice

Find out the Output (a=5,b=2,c=3)	Solution
22//8/2	ANS
5+4*5**2	ANS
5+6/2%4	ANS
a<b or 45>20 and 5<2	ANS
a<b or b<c and True or c>a	ANS
a<b or b<c and True and c>a	ANS
a<b or b<c or True and c>a	ANS

Ques	Output ??????	answer
1	7 and 0 or 2	
2	7 or 9 or 2	
3	7 and 9 or 2	
4	7 and 0 or 9	
5	-7 and -5 or 9	
6	5> "str"	
7	4 or 5> "str"	
8	A+=4/2 (A=16)	
9	A**=2//4 (A=2)	

Click on the button to view answer

a=2 , b=4, c=a, d=2

Ques	Output	Answers
1	a is b	
2	a is d	
3	a is not b	
4	4 in [1,3,4,5,7]	
5	"a" in “Divyaditya"	
6	"v" in "MANASVI"	
7	"e" in {1:"a",2:"e",3:"i",4:"o",5:"u"}	
8	"1" in {1:"a",2:"e",3:"i",4:"o",5:"u"}	
9	1 In {1:"a",2:"e",3:"i",4:"o",5:"u"}	
10	"ash" not in "natasha"	

Click on the button to view answer

PUNTCUATORS

- A **punctuator** is a token that has syntactic and semantic meaning to the compiler

‘	“	#	\			
(	)	[	]	{	}	@
,	:	.	`	=	;	

Data types in Python

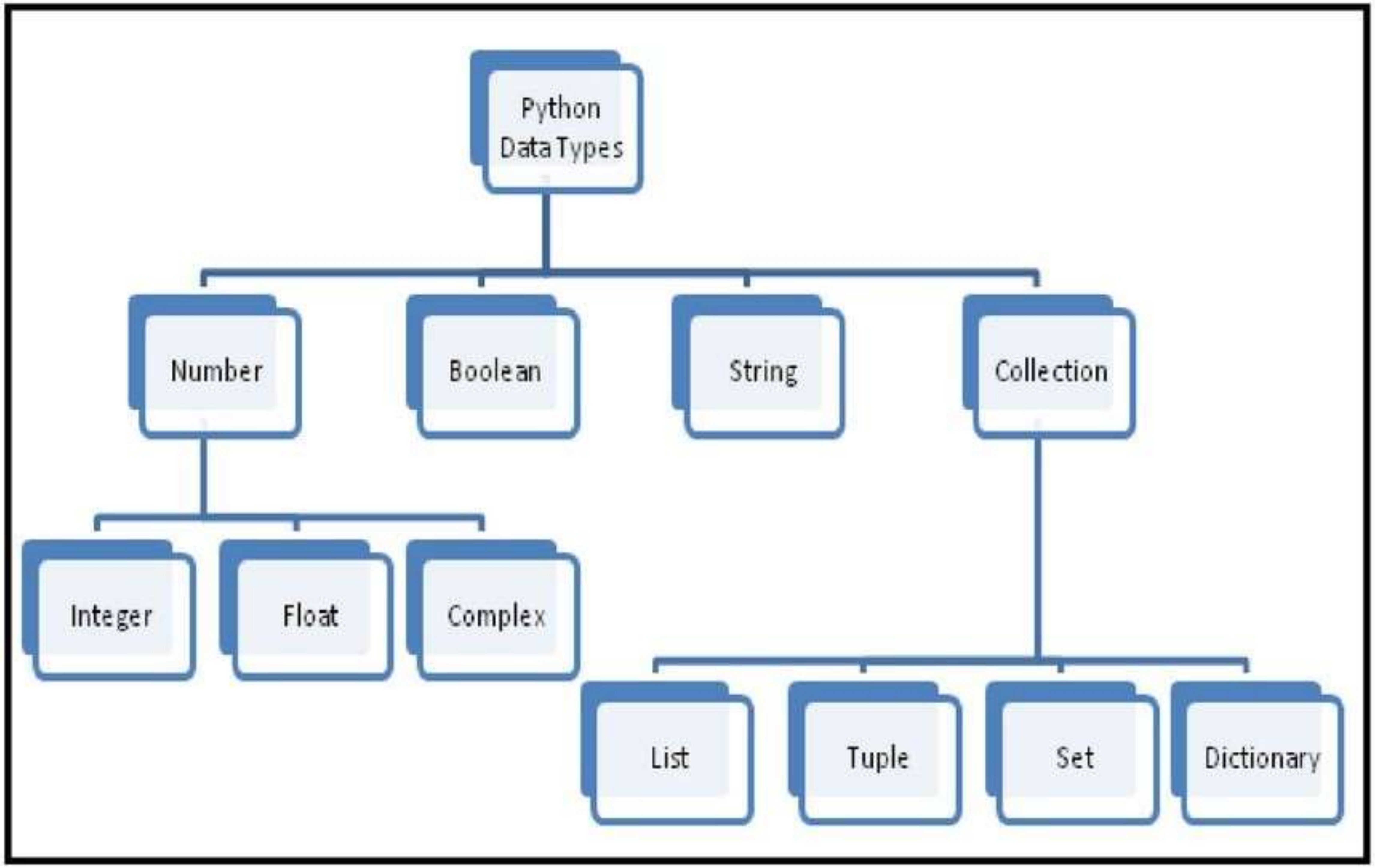
Every value in Python has a data type. Since everything is an object in Python programming, data types are actually classes, and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below

Data Types in Python

- Python Numbers
- Python List
- Python Tuple
- Python Strings
- Python Set
- Python Dictionary

Chart : Python Data Types



MUTABLE & IMMUTABLE DATA TYPES

a **mutable** object can be changed after it is created, and an **immutable** object can't.

- IMMUTABLE : Objects of built-in **types** like (int, float, bool, str, tuple, Unicode).
- MUTABLE : Objects of built-in **types** like (list, set, dict).
- Custom classes are generally **mutable**.

What are an object's **identity**, **type**, and **value**

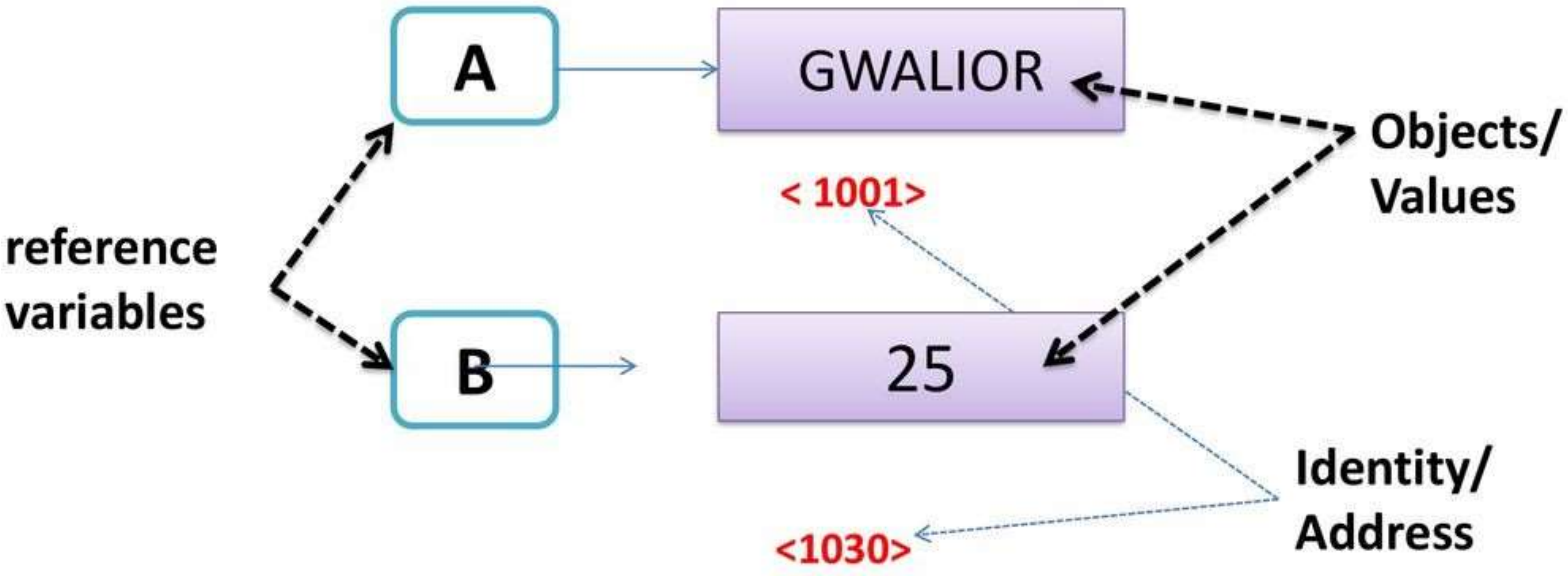
- **All the data in a Python code is treated as objects. Every object has an identity, a type, and a value.**

1. **IDENTITY : (object's address in memory)**
   identity **never changes once it has been created**
   - **Can be known by using id()**
2. **TYPE :** An object type is **unchangeable** like the identity.
   - Can be known by using **type()**
3. **VALUE :** An Object

Lets understand through an example

A="GWALIOR"

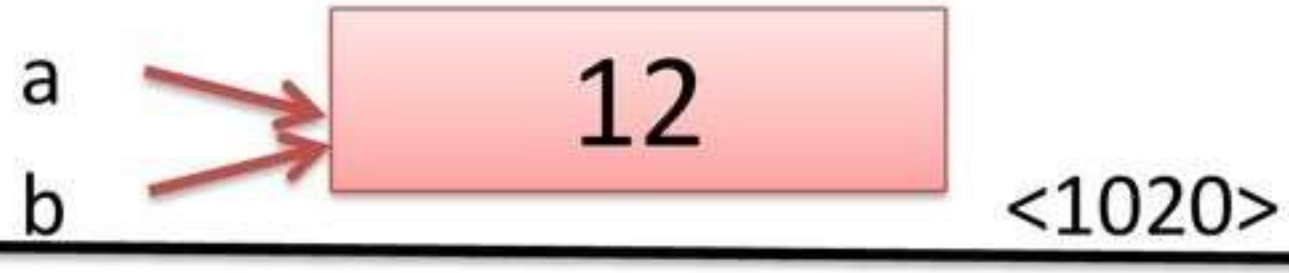
B=25



Make immutability more clear

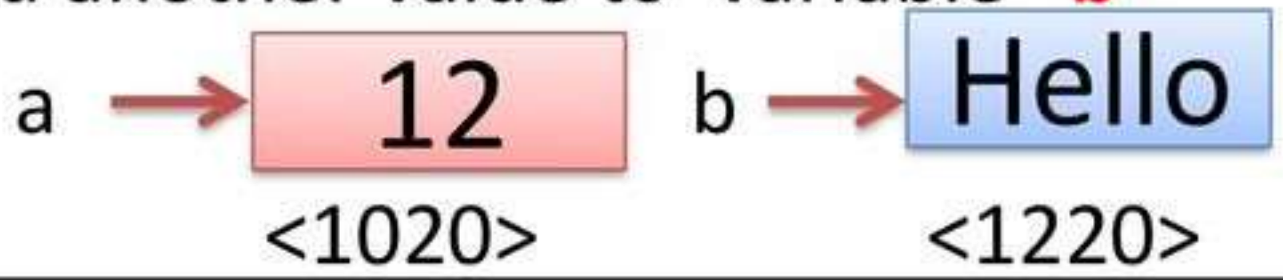
Here we have assigned same value to two different variable

```
a=12
b=12
```



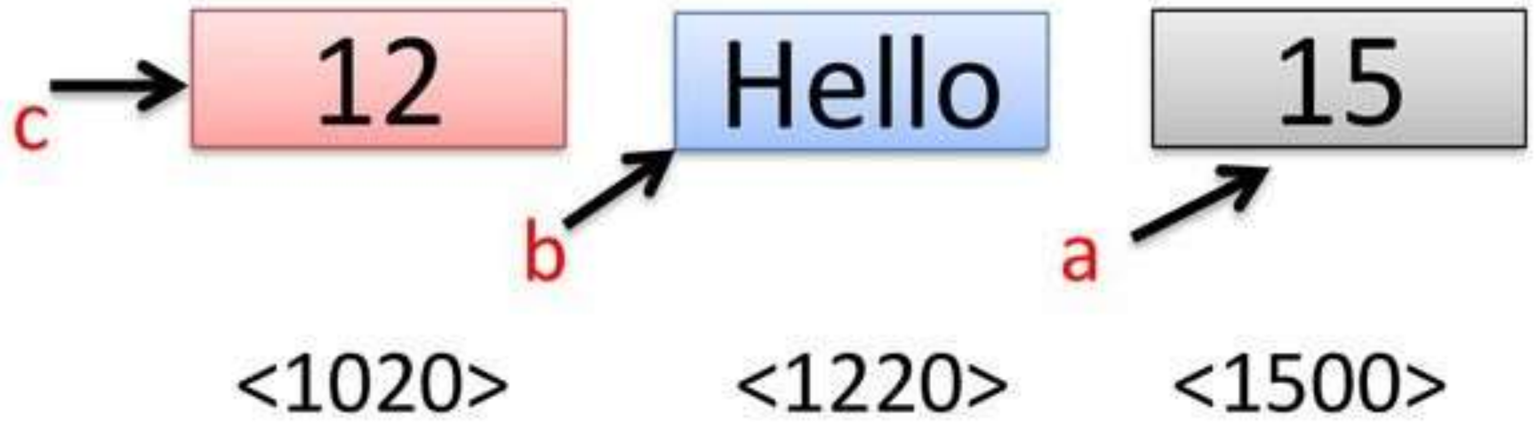
Now we have assigned another value to variable **b**

```
b="Hello"
```



Again

```
C= a
a=15
```



Creation and deletion of a Variable

```
>>> Age=24
>>> del Age
```

Dynamic typing

```
>>> A=12
>>> A=“hello”
```

Multiple assignment

```
>>> A=B=C=D=24
>>> A,B,C=1,2,3
```

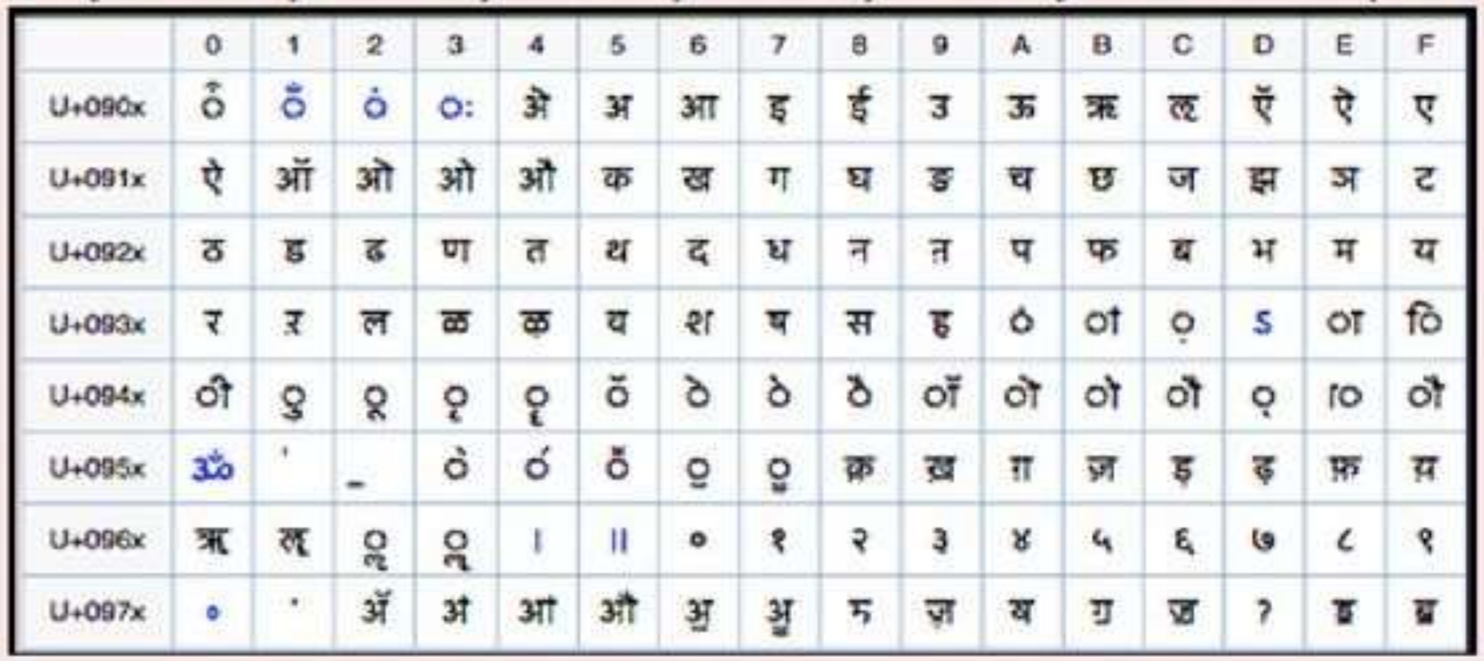
Python Data Type – Numeric

Python numeric data type is used to hold numeric values like;

Data Type	Use
Int	holds signed integers of non-limited length.
Long	Holds long integers (exists in Python 2.x, deprecated in Python 3.x).
Float	Holds floating precision numbers and its accurate up to 15 decimal places.
complex	Holds complex numbers.

Python Data Type – String

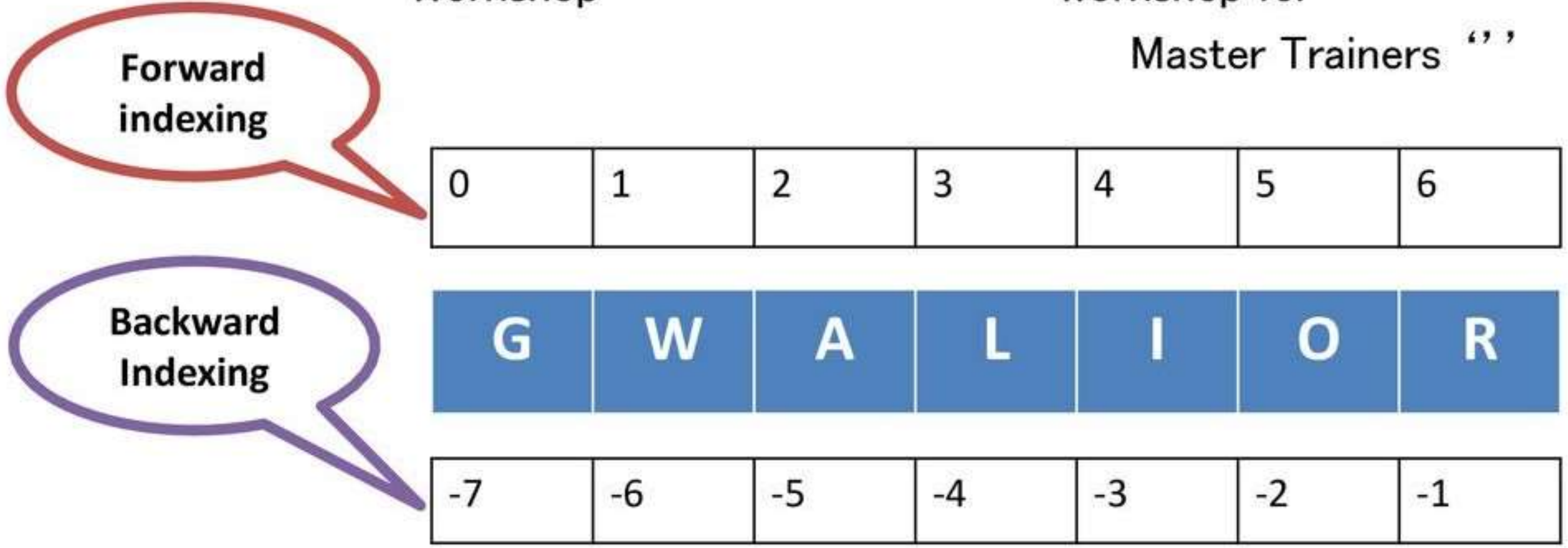
- String is a **sequence of characters**. Python supports Unicode characters. Generally strings are represented by either single or double quotes

Single Line String	"hello world"
Multi Line String	"""Gwalior Madhya Pradesh"""
Raw String	r"raw \n string" [Used when we want to have a string that contains backslash and don't want it to be treated as an escape character.]
Character	"C" [Single letter]
Unicode string	u"\u0938\u0902\u0917\u0940\u0924\u093E" will print 'संगीता' 

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+090x	◌ऀ	◌ँ	◌ं	◌ः	ऄ	अ	आ	इ	ई	उ	ऊ	ऋ	ऌ	ऍ	ऎ	ए
U+091x	ऐ	ऑ	ऒ	ओ	औ	क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
U+092x	ठ	ड	ढ	ण	त	थ	द	ध	न	ऩ	प	फ	ब	भ	म	य
U+093x	र	ऱ	ल	ळ	ऴ	व	श	ष	स	ह	◌ऺ	◌ऻ	◌़	ऽ	◌ा	◌ि
U+094x	◌ी	◌ु	◌ू	◌ृ	◌ॄ	◌ॅ	◌ॆ	◌े	◌ै	◌ॉ	◌ॊ	◌ो	◌ौ	◌्	◌ॎ	◌ॏ
U+095x	ॐ	◌॑	◌॒	◌॓	◌॔	◌ॕ	◌ॖ	◌ॗ	क़	ख़	ग़	ज़	ड़	ढ़	फ़	य़
U+096x	ॠ	ॡ	◌ॢ	◌ॣ	।	॥	०	१	२	३	४	५	६	७	८	९
U+097x	॰	ॱ	ॲ	ॳ	ॴ	ॵ	ॶ	ॷ	ॸ	ॹ	ॺ	ॻ	ॼ	ॽ	ॾ	ॿ

String Creation

- In python string can be declared in two ways.
  - Single line string
    - str = “Python Workshop” or str = ‘Python Workshop’
  - Multi line string
    - Str = ‘Python ¥
      Workshop’ **or** Text = ‘’ ’Python
      workshop for
      Master Trainers ‘’ ’



Python Data Type – List

- List is a versatile data type exclusive in Python. In a sense it is same as array in C/C++. But interesting thing about list in Python is it can simultaneously hold different type of data.

- Formally list is a ordered sequence of some data written using square brackets ([]) and separated with commas (,)

CREATION OF LIST

```
my_list = []
# empty list
```

```
my_list = [1, 2, 3]
# list of integers
```

```
my_list = [1, "Hello", 3.4]
# list with mixed data types
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Python Tuple

- Tuple is an ordered sequences of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.
- Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas.

CREATION OF TUPLE

```
# Empty tuple
my_tuple1 = ()
my_tuple2 = (1, 2, 3)
mix_tuple = (1, "Hello", 3.4)
Nested_tuple = (“Good", [4,1, 3,5], (1, 2, 3))
```

Advantages of Tuple over List

- Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- We generally use tuple for heterogeneous (different) data types and list for homogeneous (similar) data types.
- Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Python Set

- Set is an unordered collection of **unique** items.
- Set is defined by values separated by comma inside braces { }.Items in a set are not ordered

CREATING An Empty SET

```
S1=set()
```

CREATING a SET to store Subject Names

```
subjects={"Physics","Chemistry","Maths","Compute
r Science","English"}
```

- >>> S={1,2,3,4,3,5,6}
- >>> S
- {1, 2, 3, 4, 5, 6}

◈ POINTS TO REMEMBER:

- 'set' object does not support indexing
- Being an unordered collection, sets do not record element position or order of insertion
- cannot have multiple occurrences of the same element
- set cannot have mutable items

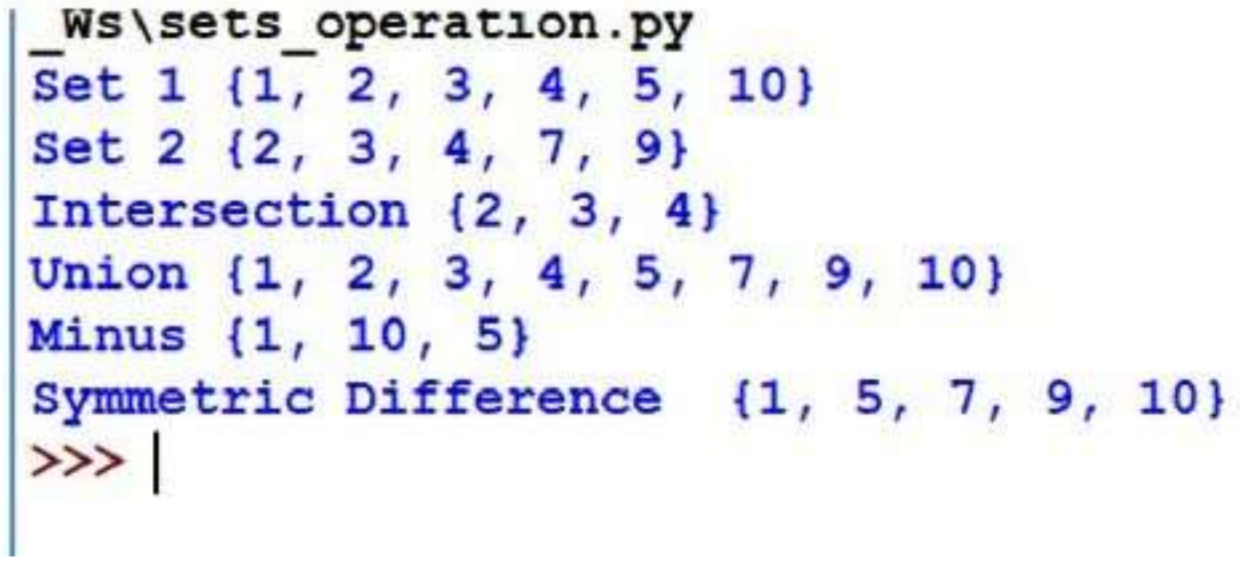
```python
a={1,2,3,4,5,10}
b={2,3,4,7,9}

print("Set 1",a)
print("Set 2",b)
inter =a&b           #Intersection
print("Intersection",inter)


uni=a|b
print("Union",uni)         #Union

minus=a-b
print("Minus",minus)       #Minus

sym=a^b             # Symmetric difference
print("Symmetric Difference ",sym)
```



```
_Ws\sets_operation.py
Set 1 {1, 2, 3, 4, 5, 10}
Set 2 {2, 3, 4, 7, 9}
Intersection {2, 3, 4}
Union {1, 2, 3, 4, 5, 7, 9, 10}
Minus {1, 10, 5}
Symmetric Difference  {1, 5, 7, 9, 10}
>>> |
```

DICTIONARY

- Python dictionary is an unordered collection of items that is changeable and indexed. dictionaries are written with curly brackets, and they have keys and values pair as data item.

```
Dict={}    # Empty Dictionary
Qtr1={1: ”Apr”,2:”May”, 3: ”Jun”}
```

Here 1,2,3 are keys and Apr , May & Jun are Values

Type Casting

- **Type Casting** is when you convert a variable value from one **type** to another.

1. If it is done by python interpreter it is called implicit type conversion
2. If it is done by user it is called Explicit type conversion

Example 1

```
a=4.0
b=5
c= a + b
```

Example 2–

```
age="25"
b=int (a)
```

Then c result of a (float type) + b (int type) will be converted into float (stored in c).

In python, for Explicit type conversion following conversion functions are use :

- int ( )
- float( )
- str( )
- complex( )
- bool( )

Syntax :

converted data = con_function (<data of another type)

How to Input Data

- **Python** has an **input** function which lets you ask a user for some text **input**. You call this function to tell the program to stop and wait for the user to key in the **data**.
- **variable=input (<“message to print”>**

Example

- **N=input(“Enter Your Name”)**

Input function read the values and convert it into string

How to print output

- **Syntax:**

  print(*objects, sep=' ', end='\n', file=sys.stdout)

Parameters:

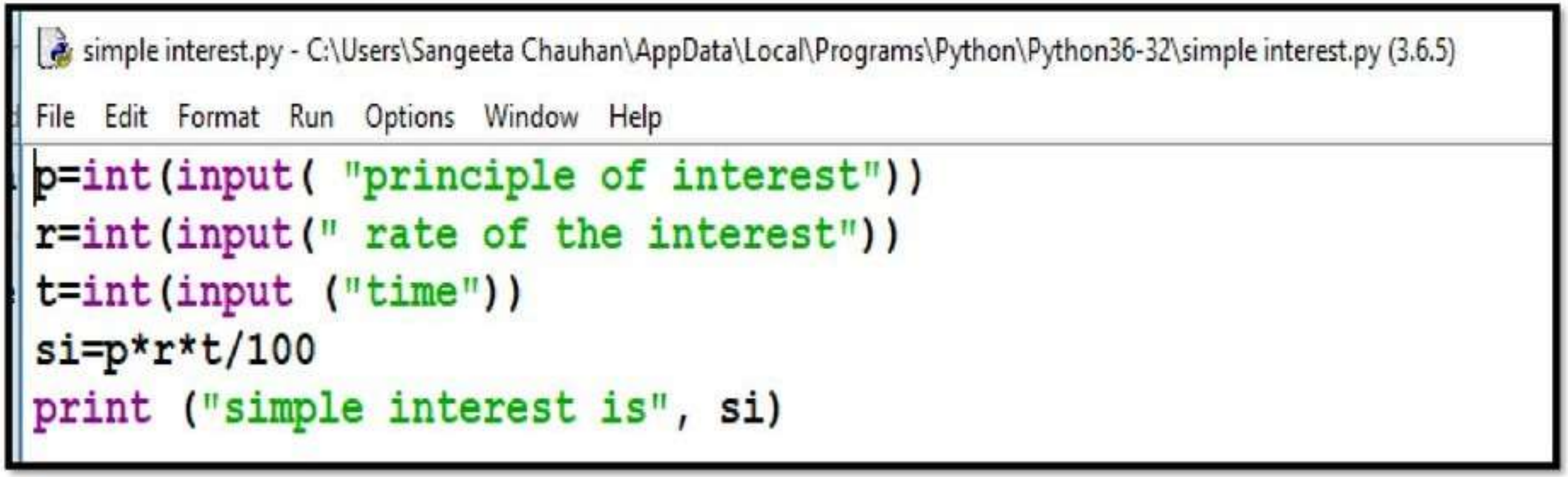
value(s) : Any value, and as many as you like. Will be converted to string before printed

sep='separator' : (Optional) Specify how to separate the objects, if there is more than one . Default is ' ' (space)

end='end': (Optional) Specify what to print at the end. Default **is** new line

file=sys.stdout : to print the message on File/standard output stream

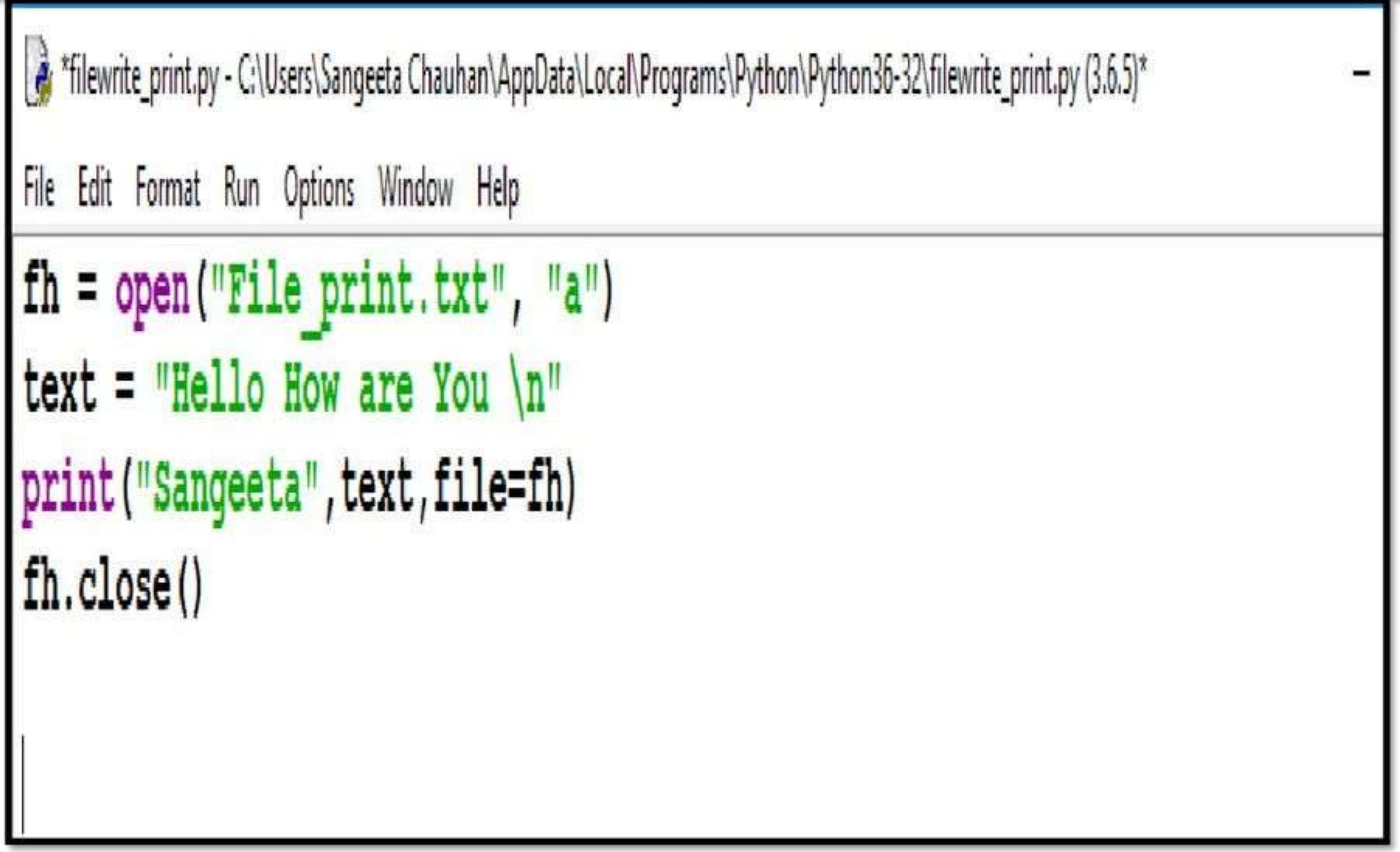
Example


simple interest.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\simple interest.py (3.6.5)
File Edit Format Run Options Window Help

```
p=int(input( "principle of interest"))
r=int(input(" rate of the interest"))
t=int(input ("time"))
si=p*r*t/100
print ("simple interest is", si)
```

In the above Example we want to read numeric values for p , r , t but **input ()** return string value hence we need to convert it into integer type using **int()**

Writing into file using print command



filewrite_print.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\filewrite_print.py (3.6.5)
File Edit Format Run Options Window Help

```
fh = open("File_print.txt", "a")
text = "Hello How are You \n"
print("Sangeeta",text,file=fh)
fh.close()
```

Types of statements

- In Python, statements are of 3 types–
  - » Empty Statements
    - pass
  - » Simple Statements (Single Statement)
    - name=input ("Enter your Name ")
    - print(name)   etc.
  - » Compound Statements
    - <Compound Statement Header>:
      <Indented Body containing multiple simple statements/compound statements>
    - Here, Header line starts with the keyword and ends at colon (:).
    - The body consists of more than one simple Python statements or compound statements.

- CONDITIONAL STATEMENTS

- <u>ITERATIVE STATEMENTS</u>

Now look at what kind of values evaluate to "True" or "False" in python. Anything that is "empty" usually evaluates to False, along with the integer 0 and the boolean value of False.

```
• for value in ('', 0, None, [], (), {}, False):
      if  value:
            print (value, "True“)
      else:
            print( value, "False" )
```

```
False
0 False
None False
[] False
() False
{} False
False False
```

Objects that are not empty evaluate to "True", along with numbers not equal to 0, and the boolean value True.

```
• for value in (' ', 2, 1, "a", [1], (3,), True):
      if value:
          print (value, "True" )
      else:
          print (value, "False“)
```

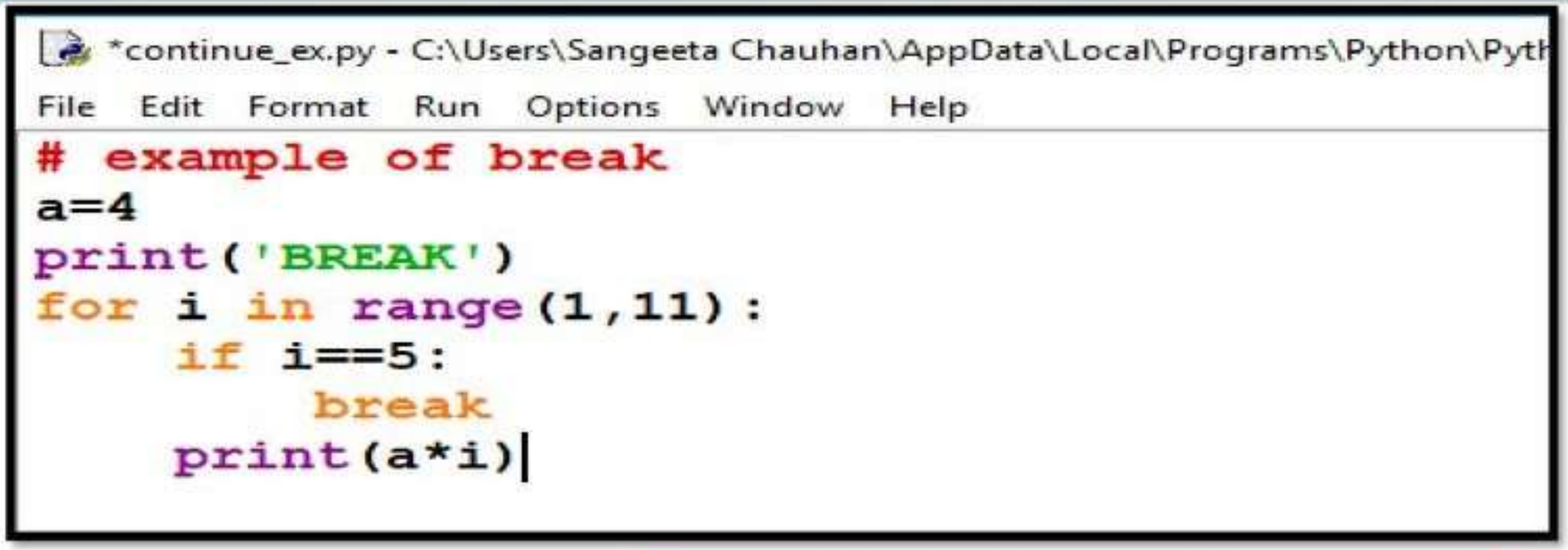
```
      True
2     True
1     True
a     True
[1]   True
(3,)  True
True  True
```

JUMP STATEMENTS

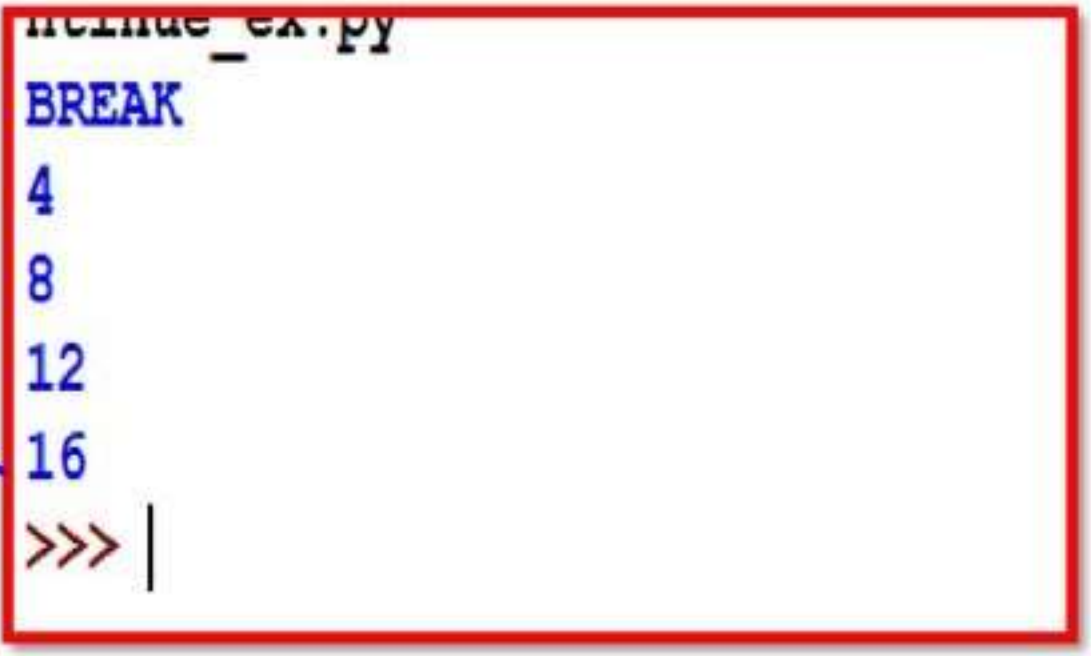
- Jump statements in python are used to modify the flow of a loop
- **Type of Jump Statements in Python:-**

1. break - to terminate the loop
2. continue -to skip all the remaining statements in the loop and move controls back to the start of the loop
3. pass-It makes a controller to pass by without executing any code

Example : break statement


*continue_ex.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Pyth
File Edit Format Run Options Window Help

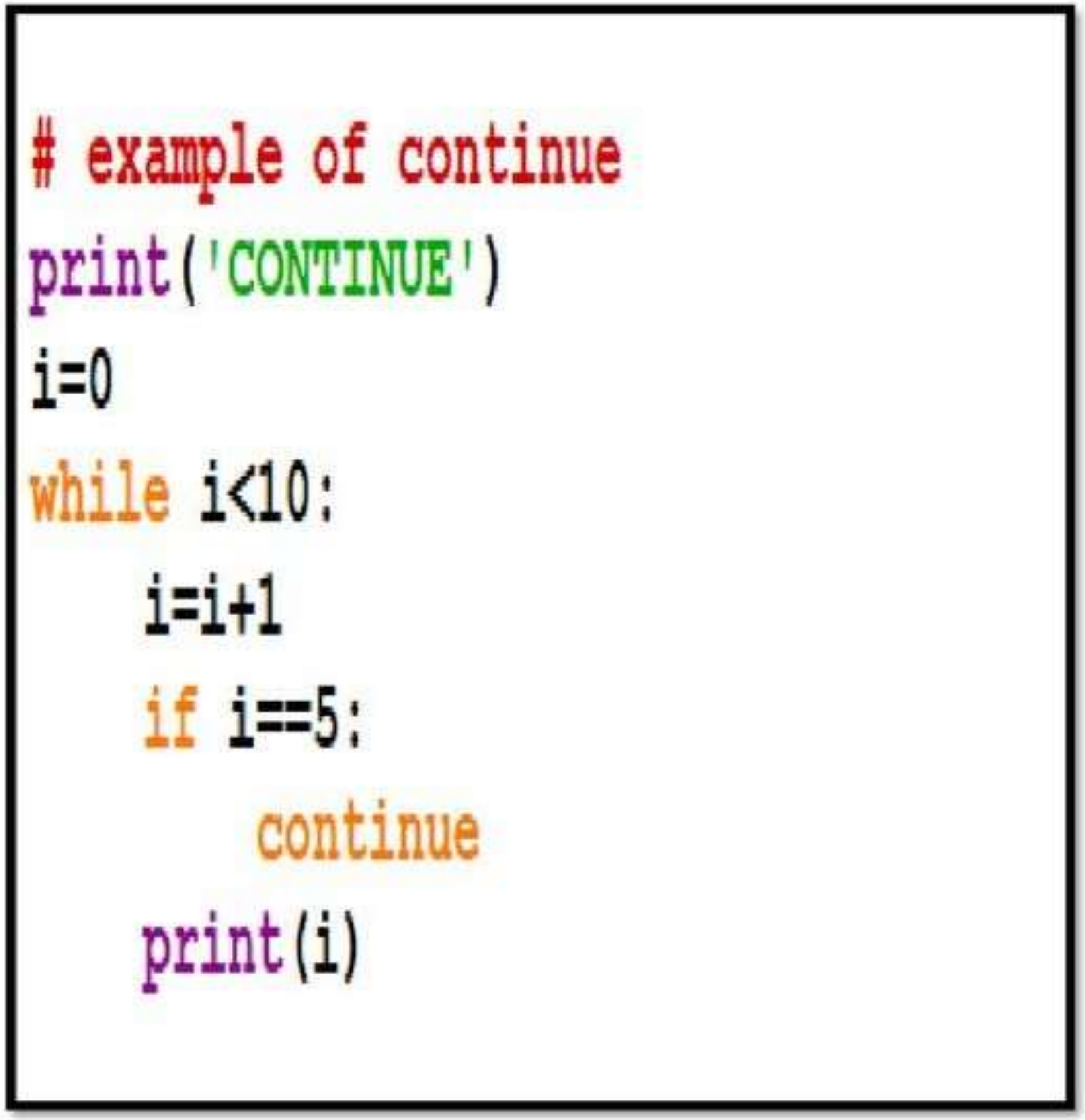
```python
# example of break
a=4
print('BREAK')
for i in range(1,11):
    if i==5:
        break
    print(a*i)
```



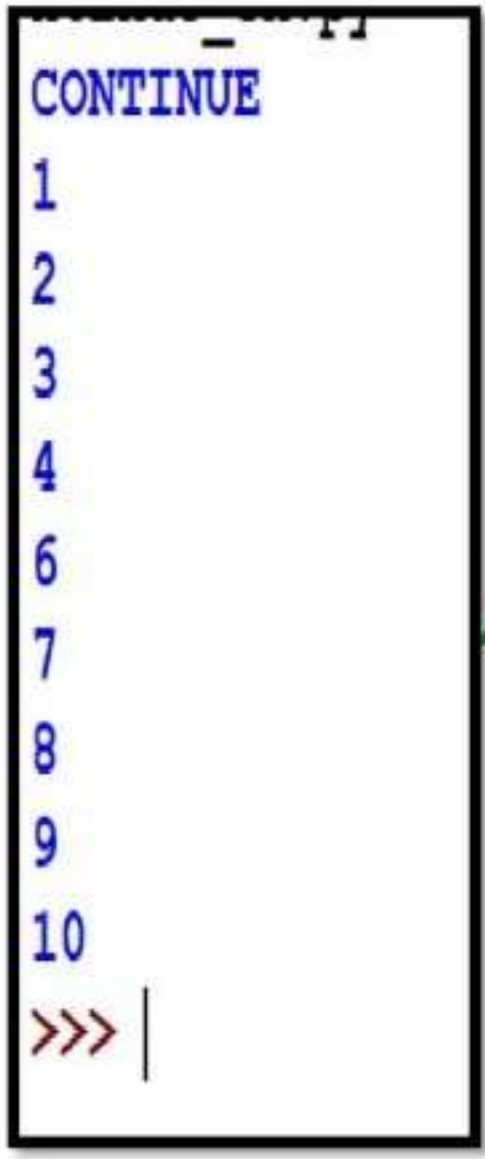
```
ntinue_ex.py
BREAK
4
8
12
16
>>> 
```

Output

Example : continue statement



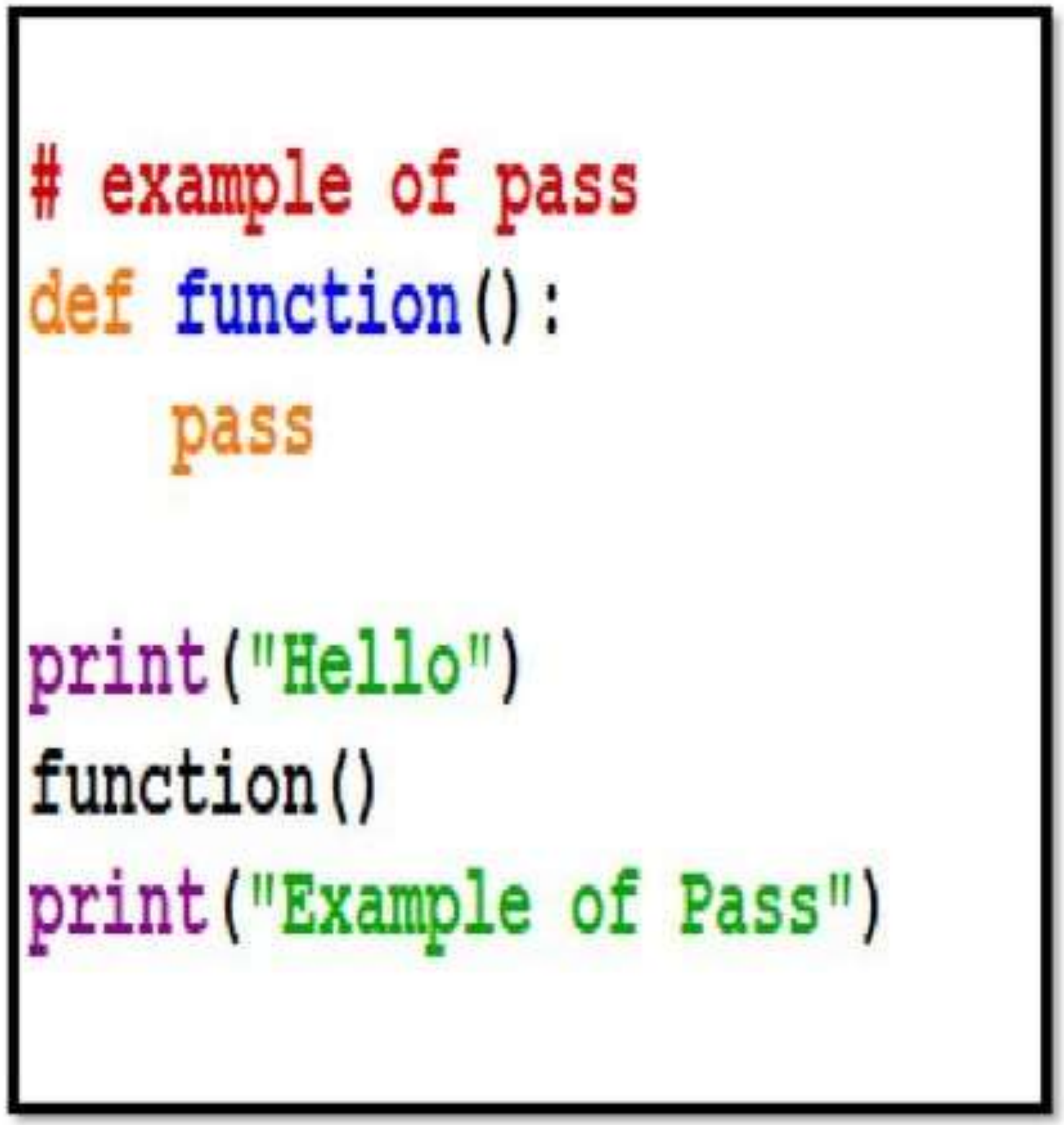
```
# example of continue
print('CONTINUE')
i=0
while i<10:
    i=i+1
    if i==5:
        continue
    print(i)
```



```
CONTINUE
1
2
3
4
6
7
8
9
10
>>> |
```

Output

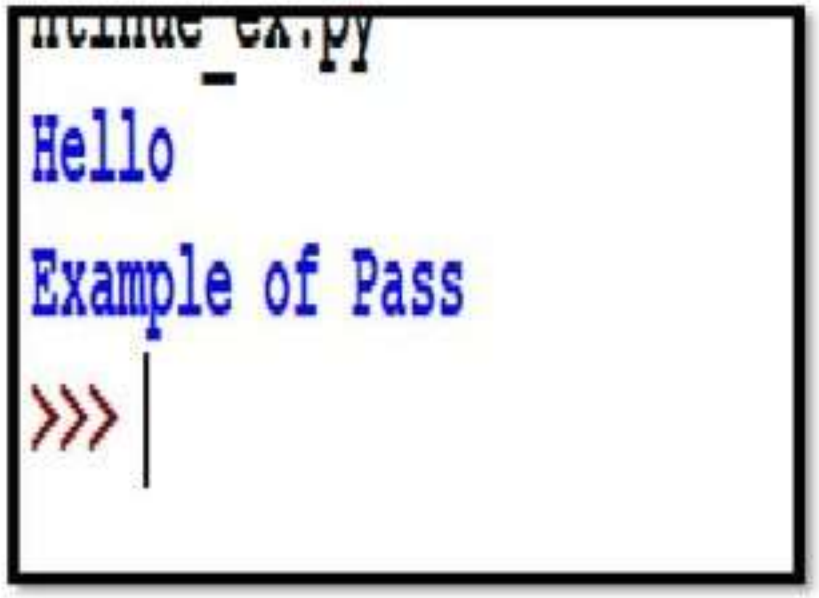
Example : pass statement



```python
# example of pass
def function():
    pass

print("Hello")
function()
print("Example of Pass")
```

Output



```
ntinue_ex.py
Hello
Example of Pass
>>>
```

the pass statement does nothing particular, but it can act as a placeholder

More to know

- **In-Place Swapping Of Two Numbers.**

```
x, y = 10, 20
print(x, y)
x, y = y, x
print(x, y)
```

- **Create a single string from all the elements in list**

```
a = [“Python", “Master", “Trainers"]
str=" ".join(a)
```

More to know...........

- **Chaining Of Comparison Operators.**

```
n = 10
result = 1 < n < 20

result = 1 > n <= 9
```

- **Python's Ternary Operator)**

```
<expr1> if  <conditional_expr> else <expr2>
```

```
category = 'minor' if age < 21 else 'adult'
```

More to know…………

- **Reversing an int through typecasting & slicing**

```
a=(str(num)[::-1])
```

- **it is permissible to write an entire if statement on one line.**

```
if (age>=18): print(" > 18"); print("An Adult "); print("Eligible to Vote");
```

Thank you!