Python Basic - Part 1

1. Variable

Variable [1] is a resevered memory location to store value, In other words, a variable in a python program gives data to the computer for processing.

Every value in python has datatype that has it own unique type.

1. String, represent letter, character, ex: "Hello World"
2. Integer, represent whole number, ex: 90
3. Float, similar with Integer, this represent a number with decimal point, ex: 82.10
4. Boolean, is one of the built-in data types provided by Python, which represents one of the two values ex: True or False.
5. List are used to store multiple items in a single variable, ex: fruits["apple", "orange", "banna"]
6. Dictionaries are used to store data values in key:value pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates, ex: employee{ "firstname": "Jack", "lastname":"Huston"}

Variable Naming Rules in Python

1. Variable name should start with letter(a-zA-Z) or underscore (_)
   Valid : age , _age , Age
   Invalid : 1age
2. In variable name, no special characters allowed other than underscore (_).
   Valid : age *, _age*
   *Invalid : age**
3. Variables are case sensitive. age and Age are different, since variable names are case sensitive.
4. Variable name can have numbers but not at the beginning.
   Example: Age1
5. Variable name should not be a Python keyword. Keywords are also called as reserved words.

Example `pass, break, continue.. etc` are reserved for special meaning in Python. So, we should not declare keyword as a variable name.

How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

```
a=100
print(a)
```

Re-declare a Variable

You can re-declare the variable even after you have declared it once.

```
a=100
print(a)
# 100

a='AECS Jaduguda'
print(a)
# AECS Jaduguda
```

Concatenate Variables

```
a='AECS'
b=1
print(a+b) # will throw error , as we cannot concatenate two different
datatypes. But:
a='AECS'
b=1 print(a+str(b)) # will display AECS1
```

2. Operators

Operators [2] are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

For example:

```
>>> 2+3
5
```

Here, `+` is the operator that performs addition. `2` and `3` are the operands and `5` is the output of the operation.

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

Operator	Meaning	Example
+	Add two operands or unary plus	x + y+ 2
-	Subtract right operand from the left or unary minus	x - y- 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

EXAMPLE 1: ARITHMETIC OPERATORS IN PYTHON

```
x = 15
y = 4

# Output: x + y = 19
print('x + y =',x+y)
```

```
# Output: x - y = 11
print('x - y =',x-y)

# Output: x * y = 60
print('x * y =',x*y)

# Output: x / y = 3.75
print('x / y =',x/y)

# Output: x // y = 3
print('x // y =',x//y)

# Output: x ** y = 50625
print('x ** y =',x**y)
```

Comparison Operators

Comparison operators are used to compare values. It returns either `True` or `False` according to the condition.

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

```
x = 10
y = 12

# Output: x > y is False
```

```
print('x > y is',x>y)

# Output: x < y is True
print('x < y is',x<y)

# Output: x == y is False
print('x == y is',x==y)

# Output: x != y is True
print('x != y is',x!=y)

# Output: x >= y is False
print('x >= y is',x>=y)

# Output: x <= y is True
print('x <= y is',x<=y)
```

Logical operators

Logical operators are the `and`, `or`, `not` operators.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

```
x = True
y = False

print('x and y is',x and y)

print('x or y is',x or y)

print('not x is',not x)
```

Assignment Operators

Assignment operators are used in Python to assign values to variables.

`x = 25` is a simple assignment operator that assigns the value 25 on the right to the variable a on the left.

There are various compound operators in Python like `x += 25` that adds to the variable and later assigns the same. It is equivalent to `x = x + 25`.

Operator	Example	Explaination	Sign state
=	x = 25	Value 25 is assigned to x	Assignment
+=	x += 25	This same as x = x + 25	Addition
-=	x -= 25	Same as x = x - 25	Substraction
*=	x *= 25	Same as x = x * 25	Multiplication
/=	x /=25	Same as x = x / 25	Division
%=	x %=25	Same as x = x % 25	Modulo
//=	x //=25	Same as x = x // 25	Floor division

3. Python Built-in Function

what is function anyway?

> *A function is **a block of organized code that is used to perform a single task**.*

There are lot of built-in function in python that can be call and execute in your code development. As a starter, there are a few functions that neccessary to be understood and utilized.

PRINT()

The Python `print()` function [3] takes in any number of parameters, and **prints them out on one line of text**.

Syntax

```
print_(object(s), sep=separator, end=end)
```

Parameter Values

Parameter	Description
object(s)	Any object, 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 ' '
end='end'	Optional. Specify what to print at the end. Dafault is '\n' (line feed)

Example:

Print more than one object:

```
print("Hello", "How are you?")
# output > Hello How are you?
```

Print 2 messages, and specify the separator:

```
print("Hello", "how are you?", sep="---")
# output > Hello---how are you?
```

Print f or f-string

string formatting [4] mechanism known as *Literal String Interpolation* or more commonly as *F-strings* (because of the leading *f* character preceding the string literal). The idea behind f-strings is to make string interpolation simpler. To create an f-string, prefix the string with the letter " f ".

```
val = 'Geeks'

print(f"{val}for{val} is a portal for {val}.")
# output > Geeks for Geeks is a portal for Geeks

name = 'Tushar'
```

```
age = 23

print(f"Hello, My name is {name} and I'm {age} years old.")
# output > Hello, My name is Tushar and I'm 23 years old.
```

Escape Characters

An escape character lets you use characters that are otherwise impossible to put into a string. An escape character consists of a backslash () followed by the character you want to add to the string. (Despite consisting of two characters, it is commonly referred to as a singular escape character.) For example, the escape character for a single quote is '. You can use this inside a string that begins and ends with single quotes. To see how escape characters work, enter the following into the interactive shell:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Python knows that since the single quote in Bob's has a backslash, it is not a single quote meant to end the string value. The escape characters ' and " let you put single quotes and double quotes inside your strings, respectively.

Escape Character	Print as
\'	Single quote
\"	Double quote
\t	Tab
\n	Newline (line break)
\\	Backslash

INPUT()

The `input()` function allows user input.

Syntax

```
input(prompt)
```

Parameter Values

Parameter	Description
prompt	A string, representing a default message before the input.

Example

Use the prompt parameter to write a message before the input and capture user response in a variable:

```
x = input('Enter your name: ')
print('Hello, ' + x)

# output > Hello Joe
```

INT()

The `int()` function returns an integer object from any number or string.

Syntax

```
int(x=0, base=10)
```

Parameter Values

Parameter	Description
x	Number or string to be converted to integer object, the default argument is zero
base	Base of the number in x. Can be 0 or 2-36.

Example

```
# integer
print("int(123) is:", int(123))
# output > int(123) is: 123

# float
```

```
print("int(123.23) is:", int(123.23))
# output > int(123.23) is: 123

# string
print("int('123') is:", int('123'))
# output > int('123') is: 123
```

FLOAT()

The `float()` method returns a floating point number from a number or a string.

Syntax

```
float(x)
```

Parameter Values

Parameter	Description
x	number or string that needs to be converted to floating point number. if it's string, thje string should be contain decimal points

Example

```
# for integers
print(float(10))
# output > 10.0

# for floats
print(float(11.22))
# output > 11.22

# for string floats
print(float("-13.33"))
# output > -13.33

# for string floats with whitespaces
print(float("     -24.45\n"))
# output > -24.45
```

```
# string float error
print(float("abc"))
# output > ValueError: could not convert string to float: 'abc'
```

STR()

The `str()` function returns the string version of the given object.

Syntax

```
str(object, encoding='utf-8', errors='strict')
```

Parameter Values

Parameter	Description
object	Required. Any object. Specifies the object to convert into a string
encoding	The encoding of the object. Default is UTF-8
errors	Specifies what to do if the encoding fails

Example

Convert a string into an integer:

```
x = str(12)
print(x)

# output > '12'
```

TYPE()

The `type()` function either returns the type of the object or returns a new type object based on the arguments passed.

Syntax

```
type(object, base, dict)
```

Parameter Values

Parameter	Description
object	Required. If only one parameter is specified, the type() function returns the type of this object
bases	Optional. specifies the base classes
dict	Optional. Specifies the namespace with the definition for the class

Example

```
a = ('apple', 'banana', 'cherry')
b = "Hello World"
c = 33

x = type(a)
y = type(b)
z = type(c)

print(x)
# output > <class 'tuple'>
print(y)
# output > <class 'str'>
print(z)
# output > <class 'int'>
```

---

1. 8-Handout.pdf (aees.gov.in)↩
2. Programiz: Learn to Code for Free↩
3. w3schools.com↩
4. f-strings in Python - GeeksforGeeks↩