Introduction to Python

Programming

Programming is a way to instruct the computer to perform various tasks.

History of Python

- Python is a open-source high level programming language.
- Python was developed by Guido van Rossum in 1991 at the National Research Institute for Mathematics & Computer Science in the Netherland.
- Python is derived from many other languages including C, C++, UNIX shell and other scripting languages.
- Python is copyrighted like Perl, Python source code is now available under the GNU General Public License (GPL).
- (i) Python 1.0 was released in 1994
  (ii) " 2.0 " " " 2000
  (iii) " 3.0 " " " 2008

Features of Python

- **High Level and Open Source Language**
  Python is a high level language and anybody can use it for free.
- **Interpreted Language**
  Python is an interpreted language as python programs are executed by an Interpreter.
- **Easy to Understand**
  Python programs are easy to understand as they have clearly defined syntax

- **Portable**
  Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

- **Databases**
  It provide interfaces to all major commercial databases.

- **GUI Programming**
  Python supports GUI application
  Ex Tkinter

- **Object Oriented**
  A programming language that can model the real world is said to be object oriented.
  Python supports both procedure-oriented and object-oriented programming which is one of the key python features.

Applications of Python

- Desktop GUIs
- Software Development
- Web and Internet Development
- Business Applications
- Youtube, Dropbox, Cinema 4D are a few globally used applications based on python.

limitations

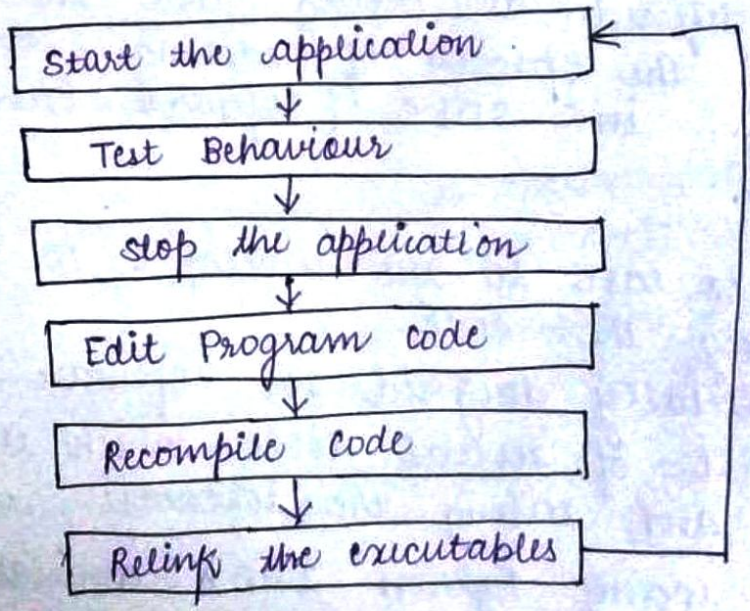
- Parallel processing can be done in Python but not elegantly as done in other languages (like Javascript)
- Being an interpreted language, Python is slow as compared to C/C++.

- Python is slower than C/C++ when it comes to computation of heavy tasks and development applications.

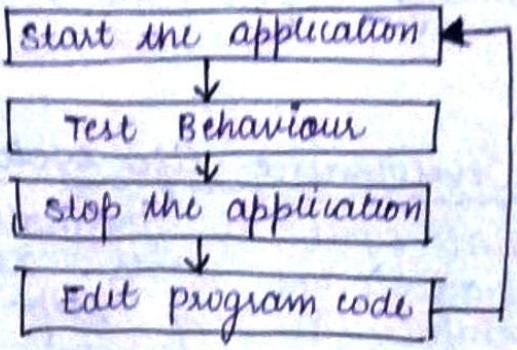
Python Program Development life cycle

- Python's development cycle is dramatically shorter than that of traditional tools. In Python, there are no compile or link step -- Python programs simply import modules at runtime and use the object they contain.
- Python programs run immediately after changes are made.
- In cases where dynamic module reloading can be used, it's even possible to change and reload parts of a running program without stopping it at all.
- Python's parser is embedded in Python-based system, it easy to modify programs at runtime.

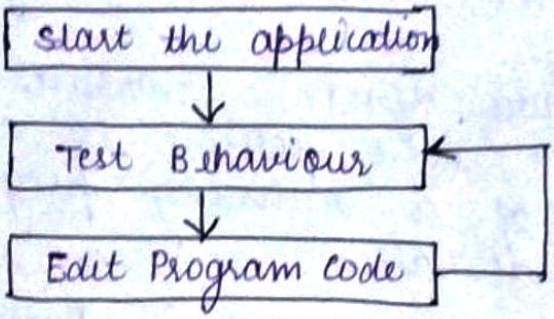
Traditional Development cycle



Python Development cycle



Python Development cycle with Module Reloading



Python IDE (Integrated Development Environment)

- IDE stands for Integrated Development Environment.
- IDE is defined as coding tool that helps to automate the process of editing, compiling and testing .etc.. in SDLC (Software Development life cycle).
- It provides ease to the developer to run, write and debug the code.
- It is specially designed for software development that consist of several tools which is used for developing and testing the software.
- There are many Python IDE's available but few of them are listed below.

ex- Pycharm, Spyder, Jupyter Notebook, VS code.

Execution Modes

There are two ways to use the python interpreter.

(a) Interactive mode

(b) Script mode

Interactive mode :-
- Interactive mode allows execution of individual statement instantaneously.
- To work with the interactive mode, we can simply type a python statement on the >>> prompt directly

```
python 3.11 shell
>>> 5+3
8
>>>
```

Script mode :- script mode allows us to write more than one instruction in a file called python source code.

```
a = int(input("a"))        Enter
b = int(input("
        Enter b : ")
c = a+b
print(c)
```

Python Keywords

- Keywords are reserved words. Each keyword has a specific meaning to the python interpreter.
- We can use a keyword in our program only for the purpose for which it has been defined.
- As Python is case sensitive keywords must be written exactly as given below:

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

In programming languages, identifiers are name used to identify a variable, fun^{n} or other entities in a program.

The rules for naming an identifier in Python are as follows:

(i) The name should begin with an uppercase or a lowercase alphabet or an underscore sign(_) This may be followed by a-z, A-Z, 0-9 or underscore (_). Ex _a, num1

(ii) Identifier can't start with a digit.

(iii) It can be of any length. However it is preferred to be short and meaningful.

(iv) It should not be a keyword reserved in python.

(v) We can't use special symbols like !, @, #, $, %, etc.

Variables

A variable in a program is uniquely identified by a name (identifier).

- Variables in python refers to an object - an item or element that is stored in memory.
- Value of variable can be string, numeric or any combination of alphanumeric characters.
- In Python, we can use assignment statement to create new variable.

Ex
```
gender = 'F'
message = 'Keep Smiling'
price = 987.9
```

comments

- comments are used to add a remark or a note in the source code.
- comments are not used by interpretes.
- There are two types of comments in Python
  (a) single line comment
  (b) Multiple line comment

(a) Single-line comment

- # symbol is used in single line comments
- Everything following the hash (#) till the end of that line is treated as a comment

Ex: # This is a single line comment.

(b) Multiple-line comment

- Triple quotes (""" """) is used for multi line comments.

```
""" This is a comment.
This is a comment, too. """
```

Note - Everything is an object

- Python treats every value or data item whether numeric, string or other type as an object.
- Every obj. in python assigned a unique identity (ID) which remains the same for the lifetime of that object.

```
num1 = 50
num2 = 40
print(id(num1))
print(id(num2))
num2 = 40 + 10
print(id(num1))
print(id(num2))
```

O/P

```
1433920576
1433923574

1433920576
1433920576
```

Python Blocks / Indentation

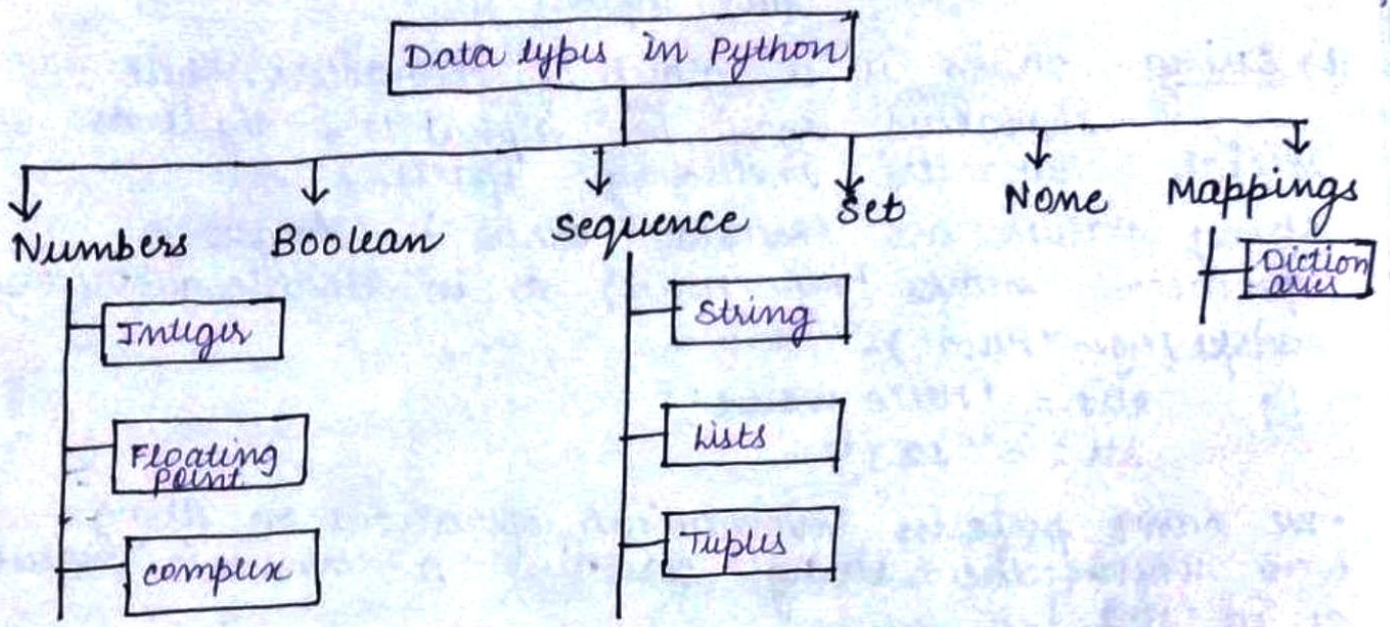
- Python does not use braces ({}) to indicate blocks of code for class and function definitions or flow control.
- Block of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

Ex

```
if True:
    print("True").
else:
    print("False)
```

Data Types in Python

Every value belongs to a specific data type in Python. Python has various standard data types that are used to define the operations possible on them & the storage method for each of them.



1. **Numbers** - Number datatype stores numerical values only. It is further classified into three different types.

Type	Description	Examples
int	integer numbers	-12, -3, 0, 152, 2
float	real or floating point numbers	-2.04, 4.0
complex	complex numbers	3+4j, 2-2j

2. **Boolean** - Boolean data type (bool) is a subtype of integer.
It is a unique data type, consisting of two constants. True and False.
- Boolean True value is non-zero, non-null & non empty.
- Boolean False is the value - zero.

3. Sequence

- A python sequence is an ordered collection of items, where each item is indexed by an integer.
- The three types of sequence data types available in python are (i) strings
  (ii) Lists
  (iii) Tuples

(i) **String** - String is a group of characters. These characters may be alphabets, digits or special characters including spaces.
String values are enclosed either in single quotation marks (e.g. 'Hello') or in double quotation marks (e.g. "Hello").

Eg
```
str1 = 'Hello World!'
str2 = "123"
```

- we can't perform numerical operations on strings even when the string contains a numeric value, as in str2.

(ii) **list**
List is a sequence of items sipvrated by commas & the items are enclosed in square brackets [].

Ex-
```
list1 = [5, 3.4, "New Delhi", "20C"]
print(list1)
```

O/P - [5, 3.4, "New Delhi", "20C"]

(iii) **Tuples**
- Tuple is a sequence of items sipvrated by commas and items are enclosed in parenthesis ().
- Once created we can't change the tuple.

Ex
```
tuple1 = (10,)
print(tuple1)
```

O/P (10,)

4. **Set**

- Set is an unordered collection of items separated by commas & the items are enclosed in curly brackets {}.
- A set is similar to list, except that it can not have duplicate entries.
- Sets are not immutable (once created, elements of a set cannot be changed).

Ex
```
set 1 = {10, 20, 30, 3.14, "New Delhi"}
set 2 = {1, 2, 1, 2}
print(set1)
print(set2)
```

O/P
```
{10, 20, 30, 3.14, "New Delhi"}
{1, 2}
```

5. **None**

- None is a special data type with a single value.
- It is used to signify the absence of value in a situation.
- None supports no special operations & it is neither same as False nor 0(zero).

Ex-
```
myVar = None
print(myVar)
print(type(myVar))
```

O/P
```
None
<class 'NoneType'>
```

6. **Mapping**

Mapping is an unordered data type in Python

(i) **Dictionary**

- Currently, there is only one standard mapping data type in Python called Dictionary.
- Dictionary holds data items in key-value pairs.

- Items in dictionary are enclosed in curly brackets {}.
- Every key is seperated from its value using a colon (:) sign.
- The key: value pairs can be accessed using the key. Keys are usually strings & their values can be any data type.

Creation of dictionary

```
dict1 = {'Fruit': 'Apple', 'climate':
        'hot', 'Price': 120}
print(dict1)
```

O/P `{'Fruit': 'Apple', 'climate': 'hot', 'price': 120}`

Accessing values from dictionary

```
print(dict1['Price'])
```

O/P 120

Note: Mutable Data Types → list, set, dictionary
Immutable Data Types → int, float, bool, string, tuples

Type conversion

- The process of converting the value of one data type to another data type is called type conversion.
- Python has two types of type conversion.
  - Implicit Type conversion
  - Explicit Type conversion

(i) Implicit Type conversion

- In implicit type conversion, python automatically converts one data type to another data type.
- This process does not need any user involvement.

Ex

```
num1 = 123
num2 = 1.23
sum = num1 + num2
print(sum)
```

O/P 124.23

(ii) **Explicit Type conversion**

In explicit type conversion, users convert the data type of an object to required data type.

<required data type> expression

Ex
```
num = 123
str = "456"
sum = num + int(str)
print(sum)
```

O/P 579

To convert b/w types, you simply use the type-name as a function.

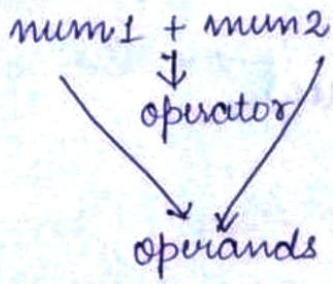
Function	Discription
int(x)	converts x to an integer
float(x)	converts x to floating point number
str(x)	converts x to string representation.
chr(x)	converts ascii value of x to character
ord(x)	returns the ASCII code of x

Example
```
>>> int('456')          O/P 456
>>> int('stringvalue')  value error
>>> float('456')        456.0
>>> str(12)             '12'
>>> chr(65)             'A'
>>> ord('A')            65
```

Operators

- An operator is used to perform specific mathematical or logical operations on values.
- The values that the operators work on are called operands.

Ex



1. Arithmetic Operators

operator	operation	Discription	Example

+	Addition	adds two numeric values	>>> num1 = 5
			>>> num2 = 6
			>>> num1+num2
			11

		also concatenate two strings on either side of operator	>>> str1 = "Hello"
			>>> str2 = "India"
			>>> str1 + str2
			"HelloIndia"

-	Subtraction	subtract the operands on the right	>>> num1 = 5
			>>> num2 = 6
			>>> num1 - num2
			-1

*	Multiplication	multiplies the numeric values	>>> num1 = 8
			>>> num2 = 4
			>>> num1 * num2
			32

		- Repeats the item on the left operator
		- if first operand is string
		- second is numeric
			>>> str1 = 'Hello'
			>>> str1 * 2
			'HelloHello'

/	Division	divides the numeric values returns quotient	>>> num1 = 8
			>>> num2 = 4
			>>> num1 / num2
			1.0

%	Modulus	divides the operands & returns remainder	>>> num1 = 13
			>>> num2 = 5
			>>> num1 %. num2
			3

//	Floor Division	• divides the numeric values
		• returns quotient after removing decimal part
			>>> num1 = 13
			>>> num2 = 4
			>>> num1 // num2
			3

**	Exponent	performs exponential (power) operation	>>> num1 = 3
			>>> num2 = 4
			>>> num1 ** num2
			81

2 Relational Operators

- Relational operates compares the value of the operands & determines the relationship among them.
- Assume the python variables

num1 = 10
num2 = 0
num3 = 10
str1 = "Good"
str2 = "Morning"

Operator	Operation	Description	Example
==	equals to	if the value of two operands are equal return True else return False	>>> num1 == num2 False >>> str1 == str2 False
!	not equal to	if the values of two operands are not equal return True else return False	>>> num1 != num2 True >>> num1 != num3 False
>	greater than	if left operand is greater return True else return False	>>> num1 > num2 True
<	less than	if the left operand is less return True else return False	>>> num1 < num3 False
>=	greater than or equal to	if the value of left operand is greater than or equals return True else return False	>>> num1 >= num2 True >>> num2 >= num3 False
<=	less than or equal to	if the value of left operand is less than or equals return True else return False	>>> num1 <= num2 False >>> num2 >= num3 True

3. Assignment Operators

Operator	Description	Example

=	assign value from right operand to left operand	>>> num1 = 2
		>>> num1
		2

+=	• adds value of left operand & right operand
	• stores in left operand
	• x += y is same as x = x+y
		>>> num1 = 10
		>>> num2 = 2
		>>> num1 += num2
		>>> num1
		12

-=	• subtracts value of left operand to right operand
	• stores in left operand
	• x -= y is same as x = x-y
		>>> num1 = 10
		>>> num2 = 2
		>>> num1 -= num2
		>>> num1
		8

*=	• multiplies values of left operand & right operand
	• stores result in left operand
		>>> num1 = 6
		>>> num2 = 3
		>>> num1 /= num2
		>>> num1
		2.0

%=	• divides values of operands
	• stores results (remainder) in left operand
		>>> num1 = 7
		>>> num2 = 3
		>>> num1 %= num2
		>>> num1
		1

`//=` • performs floor division
• and stores result in left operand

```
>>> num1 = 7
>>> num2 = 3
>>> num1 //= num2
>>> num1
2
```

`**=` • performs exponential (power)
• stores result in left operand

```
>>> num1 = 2
>>> num2 = 3
>>> num1 **= num2
>>> num1
8
```

4. Logical operators

a = True , b = False

operator	operation	Description	Example
and	Logical And	if both the operands are true returns True else False	>>> a and b False
or	Logical OR	if one of the operand is true returns True else False	>>> a or b True
not	Logical NOT	reverse the logical state of operand	>>> not (a) False

5. Bitwise Operators

Assume a = 60
b = 13

Binary = 32 16 8 4 2 1
0 1 1 1 0 0
=> 0 0 1 1 0 1

operator	operation	Description	Example
&	Binary AND	if both operand bits are 1 then copies 1 otherwise 0	>>> a & b 12
|	Binary OR	It copies a bit, if it exists in either operand	>>> a | b 61

^	Binary XOR	It copies the bit if it is set in one operand but not both.	>>> a ^ b 49
~	Binary ones complement	It is unary & has effect of flipping bits	>>> ~a 61 -(a+1) = ~a
<<	Binary left shift	shift bits towards left insert 0's on right	5 << 2 101 ⇒ 10100 ⇒ 20
>>	Binary right shift	shift bits towards right insert 0's on left	5 >> 2 101 >> 2 ⇒ 001 ⇒ 1

6. Identity Operators

- Identity operators are used to determine whether the value of a variable is of a certain type or not.
- It can also be used to determine whether two variables are referring to the same obj or not.

operator	Discription	Example

is	Evaluates True if value of operands point towards the same mem. location
	else False
		>>> num1 = 5
		>>> type(num1) is int
		True
		>>> num2 = num1
		>>> num2 is num1
		True

is not	Evaluates False if value of operands point towards the same memory location	>>> num1 is not num2 False

7. Membership Operaters

Membership operators are used to check if a value is the member of the given sequence or not.

operator	Discription	Example

in	Returns True if the var./value is found in specified sequence otherwise False	>>> a = [1,2,3]
		>>> 2 in a
		True
		>>> '2' in a
		False

not in	Returns True if the var./value is not found in specified sequence otherwise False	>>> a = [1,2,3]
		>>> 10 not in a
		True
		>>> 1 not in a
		False

Precedence of an operator

Precedence of an operator means in which order operator is evaluated. The operator that has high precedence is evaluated first.

- The following table lists all the operators from highest precedence to the lowest.

operator	Discription
**	Exponentiation
~ , + , -	complement , unary plus & unary minus
* , / , % , //	multiply, divide, modulo, floor division
+ , -	addition , subtraction
>> , <<	right bitwise & left Bitwise shift
&	Bitwise and
^ |	XOR , OR
<= , < , >= , >	comparison operators
<> , == , !=	equality operators
= , %= , /= , *= , += , -= , **=	assignment operators
is , is not	identity operators
in , not in	membership operators
AND , OR , NOT	logical operators

Example $x = 7 + 3 * 2 \Rightarrow 7 + 6 \Rightarrow 13$

Associativity

- when two or more operators have the same precedence, the associativity defines the order of operations.
- Almost all operators except the exponent (**) supports the left to right associativity

Ex → x = 4 * 7 % 3 (left to right)

x = (28) % 3

x = 1

Ex Y = 2 ** 3 ** 2 (right to left)

Y = 2 ** 9

Y = 512