INTRODUCTION OF PYTHON

The implementation of Python was started in the December 1989 by Guido Van Rossum at CWI in Netherland.
He named it after the television show Monty Python's Flying Circus.
It is a high level programming language.
Python uses interpreter for converting High Level Language into Low Level Language.
Python is case sensitive.
The Python allows Procedural & Object Oriented both approach of programming.
The default extension of Python program file is .py
It is dynamically typed.
Free and Open Source
Python provides rich library support for implementing Data Analysis, Machine Learning & IOT etc.

WHY PYTHON

Easy to learn & easy to maintain.
Fast, efficient, portable, and extendable.
GUI programming support with various database APIs.
offers many choices for web development:
Less development time than the other languages.
Largest year-on-year job demand.
Compatible with Major Platforms and Systems.
Python is widely used in scientific and numeric computing.

COMPARISON WITH OTHER LANGUAGES

Python	**C++**	**Java**

Python supports the use of FUNCTIONS and CLASSES but does not force it
For example, to print *“HELLO WORLD !”* program in Python
 	
>>> print (“HELLO WORLD !”)
	C++ requires this program as function wrapping and preceded by a preprocessor directive
	
	# include <iostream.h>
	void main()
	{
	cout <<“HELLO WORLD !” ;
	}
		In Java, situation is even worse, as all code must be inside of a class
		
		public class c1
		{
		public static void main (String [] args)
		{
		System.out.println(“HELLO WORLD !”);
		}
		}

AREAS OF PYTHON

Now a days Pyhton is the most demanding language due to its acceptance many various development areas. Few important areas are:

- ✓ Desktop and web applications.
- ✓ Mobile Applications.
- ✓ Robotics,
- ✓ Web scraping,
- ✓ Scripting,
- ✓ Artificial intelligence,
- ✓ Data analysis,
- ✓ Machine learning,
- ✓ Face detection,
- ✓ Color detection,
- ✓ 3d cad applications,
- ✓ Console-based applications,
- ✓ Audio & video-based applications,
- ✓ Enterprise applications,
- ✓ Applications for images etc.

POPULAR APPS DEVELOPED BY PYTHON

- *YOUTUBE*
- *GOOGLE*
- *FIREFOX*
- *QUORA*
- *DROPBOX*
- *INSTRAGRAM*
- *BITTORRENT*

https://www.scribd.com/document/382336941/Python-Development-Cycle

EXECUTION OF A PYTHON PROGRAM

There are various techniques for executing a Python program. These are-

1. Using IDLE
2. Using Command Prompt
3. Using IDE Environments Like Spyder, Jupyter, PyCharm etc.

1. Integrated Development and Learning Environment(IDLE)

a. It supports line by line execution of python code
   Example-
   ```
   >>> print (“HELLO WORLD !”)
   ```
b. You can create a script file and execute by pressing F5.
   Example

Program File Name : demo.py

```
print("Hello Friends");

print("Welcome in Python");
```

TOKENS

Small individual units of a program are known as tokens. It contains-

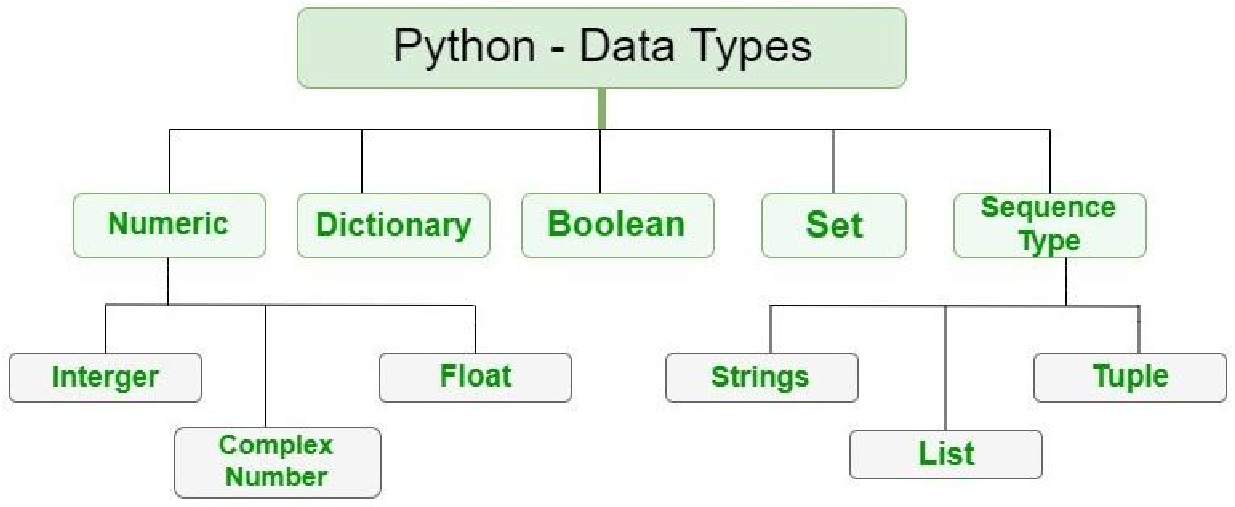
1. Identifiers
2. Keywords.
3. Constants
4. Data Type
3. Operators.

Identifiers	Identifiers are the names given to the fundamental building blocks in a program.
Keywords	They are the reserve words which implements any particular meaning in python program. There are 33 keywords in Python. Some examples are- True, None, for ,while ,break, pass etc.
Constants	They are the fixed values which did not change during execution of the program. They are also known as "***literals"***. **Literals Types:**

String literals	"Aman" , '12345'

Numeric literals	a=10
	b=1.67

Boolean literals	A Boolean literal can have any of the two values: True or False. **flag=True**
Special literals	Python contains one special literal i.e., None. **The None keyword is used to define a null variable or an object.**

Data Type	

PYTHON VARIABLES

Variable represents a memory location when we can place any value.
Python is a **dynamic typed programming language**. Here we don't need to declare the variable before its use also not need to define the type of variable.

iv=10
fv=2.5
n="Anuj"

PYTHON OPERATORS

Operators are particular symbols that are used to perform operations on operands.

Types of Operators:

Python supports the following operators:

1. Assignment Operator
2. Arithmetic Operators.
3. Relational Operators.
4. Shorthand Operators.
5. Logical Operators.
6. Membership Operators.
7. Identity Operators.
8. Bitwise Operators.

1. **Assignment Operator**
   The equals to sign(=) is known as assignment operator and it assign the value from right to left.
   Syntax
   ```
       variable=value;
       <-----------------
   ```
   Variations

   Multiple Assignment
   ```
       Example:
       x=y=z=50
   ```

   Assigning multiple values to multiple variables:
   ```
       Example:
       a,b,c=5,10,15
   ```

   Example

   ```
   a=10
   b=a
   print(a)
   print(b)
   ```

   Question- WAP for swapping the value of two variables with each other.
2. **Arithmetic Operators**
   The following table contains the arithmetic operators that are used to perform arithmetic operations.

Operators	Description
+	To perform addition
-	To perform subtraction

*	To perform multiplication
/	To perform division
%	To return remainder after division(Modulus)
//	Perform Floor division(gives integer value after division)
**	Perform exponent(raise to power)

Note- In Python 3.0, 5 / 2 will return 2.5 and 5 // 2 will return 2

Example 1.

```
a=input("First Number ")
b=input("Second Number ")
c=int(a)+int(b)
print("Sum=",c);
```

Example 2.

```
a=input("First Number ")
b=input("Second Number ")
fd=int(a)//b
md=int(a)%b
print "Floor Division=",fd
print "Modulo Division=",md
```

Questions:

1. WAP for calculating the area of rectangle.
   area=l*b
2. WAP which contains a number having three digits & write a program for reversing the digits of number.
   Example
   Input num=123
   Output=321

3. **Relational Operators**

They are used for comparing two or more than two variables & return true if condition is true otherwise return false.

Operators	**Description**
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
<>	Not equal to(Python 2.X only)

Example

```
a=10
b=20
print(a<b)
print(a==b)
print(a!=b)
```

4. **Shorthand Operators**

The following table contains the assignment operators that are used to assign values to the variables.

Operators	Description
/=	Divide and Assign
+=	Add and assign
-=	Subtract and Assign
*=	Multiply and assign
%=	Modulus and assign
**=	Exponent and assign
//=	Floor division and assign

Example

```
num=5
print(num)
num=num+2
print(num)
num+=2
print(num)
num*=5
print(num)
```

5. **Logical Operators**

They are used for taking the logical decisions.

Operators	Description
and	Logical AND
or	Logical OR
not	Logical NOT

Example:

```
a=10
b=2
c=30
print(a>b and a>c)
print(a>b or a>c)
print(c>a and c>b)
```

6. **Membership Operators**

The following table contains the membership operators.

Operators	Description
In	Returns true if a variable is in sequence of another variable, else false.
not in	Returns true if a variable is not in sequence of another variable, else false.

Example

member.py	Output

```
a=10
b=20
list=[10,20,30,40,50];
if (a in list):
    print "a is in given list"
else:
    print "a is not in given list"
if(b not in list):
    print "b is not given in list"
else:
    print "b is given in list"
```
	```
	>>>
	a is in given list
	b is given in list
	>>>
	```

7. **Identity Operators**

The following table contains the identity operators.

Operators	**Description**
Is	Returns true if identity of two operands are same, else false.
Not is	Returns true if identity of two operands are not same, else false.

Example

identity.py	**Output**

```
a=20
b=20
if( a is b):
    print  a,b have same identity
else:
    print a, b are different
b=10
if( a is not b):
    print  a,b have different identity
else:
    print a,b have same identity
```
	```
	>>>
	a,b have same identity
	a,b have different identity
	>>>
	```

OPERATOR PRECEDENCE

The operator precedence in Python are listed in the following table. It is in descending order, upper group has higher precedence than the lower ones.

Operators	Meaning
()	Parentheses
**	Exponent
+x, -x, ~x	Unary plus, Unary minus, Bitwise NOT
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction

<<, >>	Bitwise shift operators
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
==, !=, >, >=, <, <=, is, is not, in, not in	Comparisions, Identity, Membership operators
Not	Logical NOT
And	Logical AND
Or	Logical OR

Python Type Conversion and Type Casting

Type Conversion:

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

Implicit Type Conversion
Explicit Type Conversion

Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.

Example 1: Converting integer to float

```
num_int = 123
num_flo = 1.23

num_new = num_int + num_flo

print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))

print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

When we run the above program, the output will be

```
datatype of num_int: <class 'int'>
datatype of num_flo: <class 'float'>

Value of num_new: 124.23
datatype of num_new: <class 'float'>
```

Example 2: Addition of string(higher) data type and integer(lower) datatype

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))

print(num_int+num_str)
```

When we run the above program, the output will be-

```
Data type of num_int: <class 'int'>
Data type of num_str: <class 'str'>

Traceback (most recent call last):
  File "python", line 7, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Explicit Type Conversion:

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

This type conversion is also called typecasting because the user casts (change) the data type of the objects.

Syntax :

```
(required_datatype)(expression)
```

Typecasting can be done by assigning the required data type function to the expression.

Example 3: Addition of string and integer using explicit conversion

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str
```

```
print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Key Points to Remember:

Type Conversion is the conversion of object from one data type to another data type.
Implicit Type Conversion is automatically performed by the Python interpreter.
Python avoids the loss of data in Implicit Type Conversion.
Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
In Type Casting loss of data may occur as we enforce the object to specific data type.

Type Converion Python Functions

S.No.	Function & Description
1	**int(x [,base])** Converts x to an integer. The base specifies the base if x is a string.
2	**float(x)** Converts x to a floating-point number.
3	**complex(real [,imag])** Creates a complex number.
4	**str(x)** Converts object x to a string representation.
5	**repr(x)** Converts object x to an expression string.
6	**eval(str)** Evaluates a string and returns an object.
7	**tuple(s)** Converts s to a tuple.
8	**list(s)** Converts s to a list.
9	**set(s)** Converts s to a set.
10	**dict(d)** Creates a dictionary. d must be a sequence of (key,value) tuples.
11	**frozenset(s)** Converts s to a frozen set.
12	**chr(x)** Converts an integer to a character.
13	**unichr(x)** Converts an integer to a Unicode character.
14	**ord(x)** Converts a single character to its integer value.

15	**hex(x)** Converts an integer to a hexadecimal string.
16	**oct(x)** Converts an integer to an octal string.