Python Datatypes, Variables, Input

Output Statements

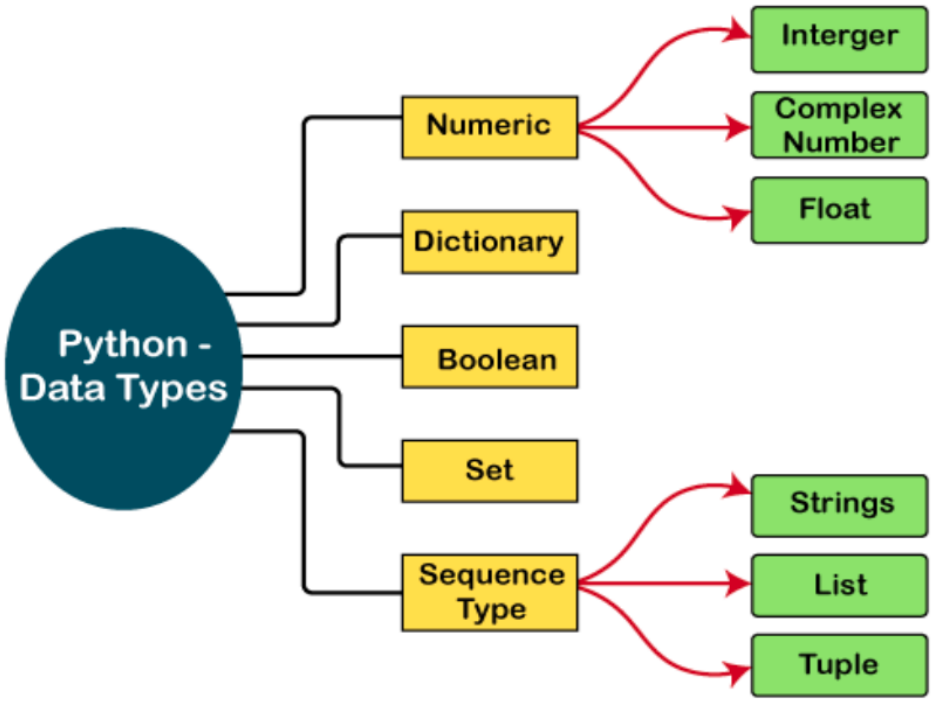
PYTHON INTRODUCTION

- **Python** is a very popular **general-purpose interpreted, interactive, and high-level programming language.**
- Works on **different platforms** (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a **simple syntax** similar to the English language.
- Allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a **procedural way, an object-oriented way or a functional way.**

WHY PYTHON?

- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

PYTHON DATA TYPES



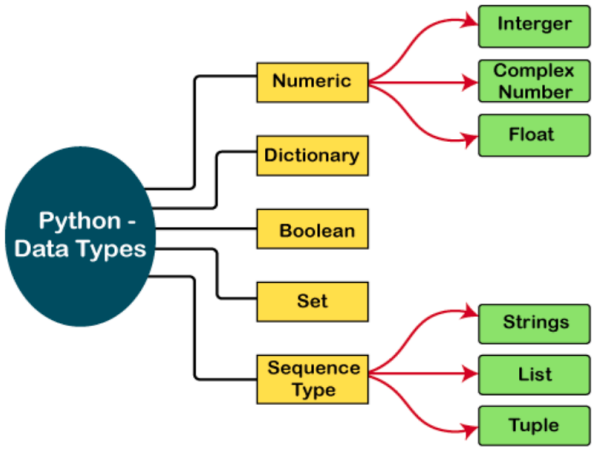
DATA TYPES

- Variables can hold values, and every value has a data-type.
- Python is a dynamically typed language; hence **we do not need to define the type of the variable while declaring it.**
- The interpreter implicitly binds the value with its type. Eg. **a = 5**.
- The variable **a** holds integer value five and we did not define its
- type.
- Python interpreter will automatically interpret variables **a** as an integer type.
- Python provides us the **type()** function, which returns the type of the variable passed.

1) NUMBERS

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as **Python int, Python float and Python complex** classes in Python.

Number stores numeric values. **Numeric Types: int, float, complex**



Int – Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length such as 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer.

Float – Float, or "floating point number" is a number, positive or negative, containing one or more decimals like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.

complex – A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc. They are written with a "j" as the imaginary part. Python creates Number objects when a number is assigned to a variable.

SAMPLE OUTPUT - NUMBERS

INT

```
a = 5
print(a)
print("The type of a", type(a))
```

```
5
The type of a <class 'int'>
```

FLOAT

```
b = 40.5
print(b)
print("The type of b", type(b))
```

```
40.5
The type of b <class 'float'>
```

COMPLEX

```
c = 1+3j
print(c)
print("The type of c", type(c))
```

```
(1+3j)
The type of c <class 'complex'>
```

2) SEQUENCE TYPES (STRING)

The string can be defined as the sequence of characters represented in the quotation marks.

In Python, we can use single, double, or triple quotes to define a string.

```
#You can use double or single quotes:

print("Hello")
print('Hello')
```

```
Hello
Hello
```

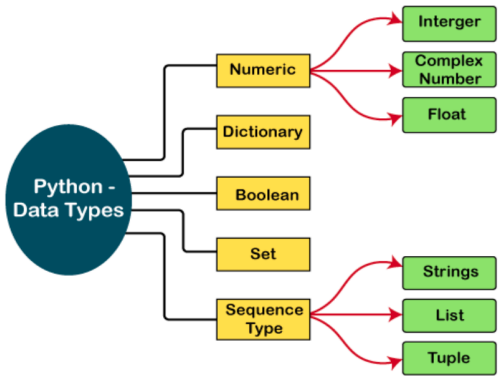
```
a = "Good Morning"
print(a)
```

```
Good Morning
```

Multiline Strings

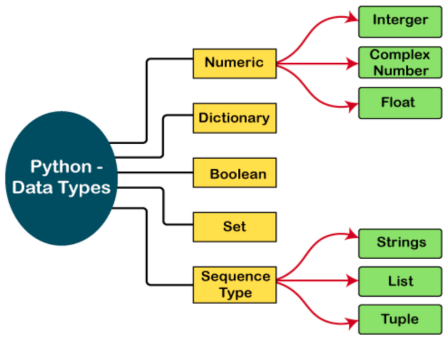
```
a = """Good Morning Friends
Welcome to Python
Tutorial."""
print(a)
```

```
Good Morning Friends
Welcome to Python
Tutorial.
```



2) SEQUENCE TYPE (LIST))

- **Lists** contain data of different types.
- List is **ordered (**it means that the items have a defined order, and that order will not change.)
- A list is **mutable or changeable** i.e we can make any changes in the list. (meaning that we can change, add or remove items after the list has been created.)
- **Allows duplicates**
- A list is a **non-homogeneous data structure** that stores the elements in columns of a single row or multiple rows.
- The list can be represented by [ ].
- The items stored in the list are separated with a comma (,).
- The list allows **nested among all.**
- Creating an empty list l=[]



- Python knows a number of *compound* data types, which are used to group together other values. The most versatile is the *list*, which can be written as a sequence of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

```
fruits=["apple", "banana", "cherry"]
print(fruits)

odd_numbers=[1,3,5,7]
print(odd_numbers)


print(fruits[2])
print(odd_numbers[1])
```

OUTPUT

```
['apple', 'banana', 'cherry']
[1, 3, 5, 7]


cherry
3
```

NESTED LIST

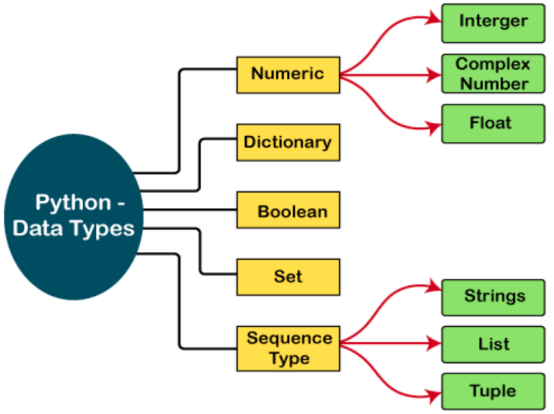
```
nested_list = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

print(nested_list)

print(nested_list[0])
```

OUTPUT

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
```

2) SEQUENCE TYPE (TUPLE))

- A tuple is a collection which is **ordered, unchangeable, allows duplicates.**
- In Python tuples are written with **round brackets.**
- Objects separated by commas. In some ways, a tuple is similar to a list in terms of indexing, nested objects, and repetition but a tuple is **immutable**, unlike lists that are mutable.
- Tuples are used when data should not be changed, such as **e.g., coordinates or constants.**



TUPLE EXAMPLE

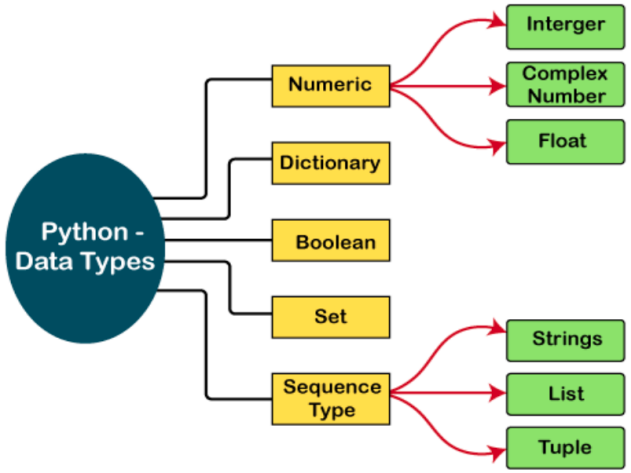
```
marks = (70, 80, 90)

print(marks)
print(marks[1])
```

OUTPUT

```
(70, 80, 90)
80
```

3) SETS

- Sets are used to store multiple items in a single variable.
- A set is a collection which is **unordered, unindexed and do not allow duplicates**
- In Python, sets are written with curly brackets.
- To determine how many items a set has, use the len() function.



SETS EXAMPLE

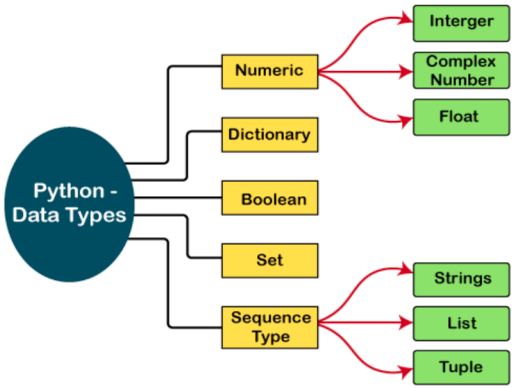
```python
class_A_students = {"Alice", "Bob", "Charlie", "David"}
class_B_students = {"Charlie", "David", "Eve", "Frank"}

# Students in either Class A or Class B (Union)
all_students = class_A_students.union(class_B_students)
print("All Students:", all_students)

# Students in both Class A and Class B (Intersection)
students_in_both = class_A_students.intersection(class_B_students)
print("Students in both classes:", students_in_both)
```

OUTPUT

```
All Students: {'Charlie', 'Bob', 'Frank', 'Alice', 'David', 'Eve'}

Students in both classes: {'Charlie', 'David'}
```

4) DICTIONARY

- 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.**
- Dictionaries cannot have two items with the same key:
- To determine how many items a dictionary has, use the len() function:



DICTIONARY EXAMPLE

```python
book_info = {
    "title": "The Hitchhiker's Guide to the Galaxy",
    "author": "Douglas Adams",
    "year_published": 1979,
    "genre": "Science Fiction",
    "pages": 193
}

# Accessing values using keys
print("BOOK TITLE IS",book_info['title'])
print("BOOK INFO IS", book_info['author'])

# Adding a new key-value pair
book_info["publisher"] = "Pan Books"
print(book_info['publisher'])
```

OUTPUT

```
BOOK TITLE IS The Hitchhiker's Guide to the Galaxy
BOOK INFO IS Douglas Adams
Pan Books
```

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

5) BOOLEAN

- Booleans represent one of two values: **True or False.**

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

- The bool() function allows you to evaluate any value, and give you True or False in return.

```
True
False
False
```

- Almost any value is evaluated to **True** if it has **some sort of content.**
- Any **string** is **True, except empty strings.**
- Any **number** is **True, except 0.**
- Any list, tuple, set, and dictionary are True, except empty ones.

```
print(bool("Hello"))
print(bool(15))
print(bool([]))
```

```
True
True
False
```

Comparison of Python Data Structures: List vs Tuple vs Set vs Dictionary

Feature	List	Tuple	Set	Dictionary
Syntax	`[ ]`	`( )`	`{ }`	`{ key: value }`
Ordered?	Yes	Yes	No	Yes (Python 3.7+)
Mutable?	Yes	No	Yes	Yes
Allows duplicate?	Yes	Yes	No	Keys: No, Values: Yes
Indexing	Yes (via index)	Yes (via index)	No (unordered)	Keys instead of index
Best use case	Ordered collection, changeable	Fixed ordered collection	Unique items, fast membership	Key-value mapping
Example	`[1, 2, 3]`	`(1, 2, 3)`	`{1, 2, 3}`	`{'a': 1, 'b': 2}`

DATA TYPES IN PYTHON

Text Type:	str
Numeric Types:	int, float, complex, BOOL
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Mutable data types :

- Lists
- Dictionaries
- Sets

Immutable data types in Python:

- Integers
- Floating-Point numbers
- Booleans
- Strings
- Tuples

Input and Output in Python

TAKING INPUT FROM USER

Python provides an **input()** function to accept input from the user.

Syntax:

input('prompt')

where, prompt is a string that is displayed on the string at the time of taking input.

Example:

```
name = input("Enter your name: ")
print(name)
```

```
Enter your name: PRIYA
PRIYA
```

TAKING INPUT FROM USER (ctd.)

- By default **input() function takes the user's input in a string.**
- So, to take the input in the form of int, you need to use int() along with input function - **Explicit type casting**

```
a=input("enter integer number")
b=input("enter float number")
c=input("enter string")
print(type(a))
print(type(b))
print(type(c))
```

OUTPUT

```
enter integer number34
enter float number56.1
enter stringamrita
<class 'str'>
<class 'str'>
<class 'str'>
```

TAKING INPUT FROM USER (ctd.)

Python Script

```
a=int(input("enter integer number"))
b=float(input("enter float number"))
c=input("enter string")
print(type(a))
print(type(b))
print(type(c))
```

OUTPUT

```
enter float number 34.78
enter string amrita
<class 'int'>
<class 'float'>
<class 'str'>
```

```
# Taking input from the user as integer
num = int(input("Enter a number: "))

add = num + 1

# Output
print(add)
```

```
Enter a number: 6
7
```

DISPLAYING OUTPUT

Python provides the **print()** function to display output to the console.

Python Script

```
# Python program to demonstrate
# print() method
print("GFG")
print('G','F','G')
print('G', 'F', 'G', sep ="*")
print("Python", end = '@')
print("GeeksforGeeks")
a=55
print("the value of a is",a)
b=10
print("the value of a is ",a," ",           "The value of b is ", b)
```

OUTPUT

```
GFG
G F G
G*F*G
Python@GeeksforGeeks
the value of a is 55
the value of a is  55   The value of b is  10
```

PYTHON COMMENTS

- Comments can be used to **explain Python code**, to **make the code more readable**, to **prevent execution when testing code**.
- Python does **not** have a built-in multiline comment like /* */ in C.
- Comments starts with a **#**, and Python will ignore them:

```
#This Line is a Comment
print("Hello, World!")
```

MULTILINE COMMENTS

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
'''
This is also
a multiline comment
'''
print("Hello, World!")
```

VARIABLES

VARIABLES

- Variables are **containers for storing data values.**
- Python has **no command for declaring a variable.**
- Python **does not require explicit declaration of a variable's type** before assigning a value.
- A variable is created the moment you first assign a value to it.

```
age = 21
colour = "lilac”
total_score = 90
```

VARIABLE NAMES

- A variable can have a **short name** (like x and y) or a **more descriptive name** (age, carname, total_volume).

Rules for Python variables:

- A variable name must **start with a letter or the underscore character**
- A variable name **cannot start with a number**
- A variable name can only contain **alpha-numeric characters and underscores (A-z,0-9, and _)**
- Variable names are **case-sensitive** (age, Age and AGE are three different variables)

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

Assign Value to Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Apple", "Pomegranante", "Grapes"

print(x)
print(y)
print(z)
```

```
Apple
Pomegranante
Grapes
```

ASSIGN SAME VALUE TO MULIPLE VARIABLE

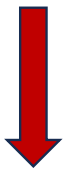
```
x = y = z = "Orange"

print(x)
print(y)
print(z)
```

OUTPUT VARIABLES

The Python print statement is often used to output variables.

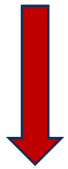
To combine both text and a variable, Python uses the + character:

```
x = "Python is"
y= "awesome"
z=x+y
print(x+y)
```



```
Python is awesome
```

```
x = 5
y = "John"
print(x + y)
```



```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

OUTPUT VARIABLES

```
x = 5
y = 10
print(x + y)
```

OUTPUT??

CASTING

- **Casting** in Python, also known as **type conversion**, refers to the process of **changing a variable's data type from one type to another**.
- This is often necessary when performing operations that require specific data types or when **data needs to be presented in a different format.**

✓**int():** Converts a value to an integer. It can take an integer literal, a float literal (truncating decimals), or a string literal representing a whole number.

✓**float():** Converts a value to a float. It can take an integer literal, a float literal, or a string literal representing a number.

✓**str():** Converts a value to a string. It can take various data types and convert them into their string representation.

✓**bool():** Converts a value to a boolean (True or False). Generally, non-zero numbers and non-empty sequences (strings, lists, etc.) convert to True, while zero and empty sequences convert to False.

PYTHON CASTING EXAMPLE

```
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
1
2
3
```

```
x = float(1)
y = float(2.8)
z = float("3")
w = float("4.7")
print(x)
print(y)
print(z)
print(w)
```

```
1.0
2.8
3.0
4.7
```

```
x = str("s1")
y = str(2)
z = str(3.0)
print(x)
print(y)
print(z)
```

```
s1
2
3.0
```

Operators in Python

OPERATORS IN PYTHON

Operators are used to perform operations on variables and values.

Operators in Python

Operators	Type
+, -, *, /, %	Arithmetic operator
<, <=, >, >=, ==, !=	Relational operator
AND, OR, NOT	Logical operator
&, |, <<, >>, -, ^	Bitwise operator
=, +=, -=, *=, %=	Assignment operator

1) ARITHMETIC OPERATOR

Arithmetic operators are used with numeric values to perform common mathematical operations:

```python
# Variables
a = 15
b = 4

# Addition
print("Addition:", a + b)

# Subtraction
print("Subtraction:", a - b)

# Multiplication
print("Multiplication:", a * b)

# Division
print("Division:", a / b)

# Floor Division
print("Floor Division:", a // b)

# Modulus
print("Modulus:", a % b)

# Exponentiation
print("Exponentiation:", a ** b)
```

```
Addition: 19
Subtraction: 11
Multiplication: 60
Division: 3.75
Floor Division: 3
Modulus: 3
Exponentiation: 50625
```

```
print("4//3: ",4//3);
print("4.0//3.0: ",4.0//3.0);
print("12//5:",12//5);
print("-22//6 :",-22//6);
print("22//-6: ",22//-6);
print("-12//8:",-12//8);
print("-23//4 :",-23//4);
print("-23//-4 :",-23//-4);
```

```
4//3:  1
4.0//3.0:  1.0
12//5: 2
-22//6 : -4
22//-6:  -4
-12//8: -2
-23//4 : -6
-23//-4 : 5
```

Note:
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)

2) RELATIONAL OR COMPARISON OPERATORS

- **Relational Operator is used to compares two values.**
- It either returns **True** or **False** according to the condition.

Operator	Name
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
a = 13
b = 33

print(a > b)
print(a < b)
print(a == b)
print(a != b)
print(a >= b)
print(a <= b)
```

```
False
True
False
True
False
True
```

3) LOGICAL OPERATOR

Logical operators are used to combine conditional statements

Logical AND, **Logical OR** and **Logical NOT** operations. It is used to combine conditional statements.

```
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)
```

Output

```
x and y is False
x or y is True
not x is False
```

4) BITWISE OPERATOR

- Python bitwise operators are used to perform **bitwise calculations on integers**.
- The integers are converted into binary format and then **operations are performed bit by bit,** hence the name **bitwise operators.**
- Python bitwise operators work on integers only and the final output is returned in the decimal format.

- ✓Bitwise AND
- ✓Bitwise OR
- ✓Bitwise NOT
- ✓Bitwise XOR
- ✓Bitwise Shift

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

EXAMPLE – BITWISE OPERATOR

```
a = 10
b = 4
print(a & b)
print(a | b)
print(~a)
print(a ^ b)
print(a >> 2)
print(a << 2)
```

```
0
14
-11
14
2
40
```

WORING OF AND OPERATOR

```
a = 0000 1010
b = 0000 0100
---------------
AND = 0000 0000   → decimal 0
```

5) IDENTITY OPERATOR

- Identity operators are : **is** and **is not**
- They are used to check if two values (or variables) are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

# Output: False
print(x1 is not y1)

# Output: True
print(x2 is y2)

# Output: False
print(x3 is y3)
```

Output

```
False
True
False
```

Explanation:

Here, we see that `x1` and `y1` are integers of the same values, so they are equal as well as identical. Same is the case with `x2` and `y2` (strings).

But `x3` and `y3` are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal.

6) MEMBERSHIP OPERATOR

Membership operators are used to test if a sequence is presented in an object:

Operator	Description
in	Returns True if a sequence with the specified value is present in the object
not in	Returns True if a sequence with the specified value is not present in the object

```
x = 'Hello world'
y = {1:'a',2:'b'}

# Output: True
print('H' in x)

# Output: True
print('hello' not in x)

# Output: True
print(1 in y)

# Output: False
print('a' in y)
```

```
Output

True
True
True
False
```

Arithmetic operations on strings

+ operator (for concatenation):

When you use + on strings, it will concatenate them together.

```
print('hello' + 'world')
```
Output:
```
helloworld
```

* Operator (for repetition):

When you multiply a string by an integer, n, the string will be repeated n times:

```
print('hello' * 3)
```
Output:
```
hellohellohello
```

Note:

If you place two strings next to each other, they will also be concatenated:
```
print('hello' 'world')
```
Output:
```
helloworld
```

DISCUSSION

An **identifier** is the name you give to things you create in your program: (variable name, function name, class name, module name, object name)

1) What is the maximum possible length of an identifier in Python?

a) 31 characters

b) 63 characters

c) 79 characters

d) none of the mentioned

Answer: d (Explanation: Identifiers can be of any length)

2) Which of the following is invalid?

a) _a = 1

b) __a = 1

c) __str__ = 1

d) none of the mentioned

Answer: d

3) What will be the output of the following program on execution?

```
if False:
        print ("inside if block")
elif True:
        Print ("inside elif block")
else:
        print ("inside else block")
```

A. inside if block
B. inside elif block
C. inside else block
D. Error

Ans : B

4) What will be the output of following Python code snippet?

```
str1="012"
num1=2
num2=0
for i in range(4):
num1+=2
for j in range(len(str1)):
num2=num2+num1
num3=num2%int(str1)
print(num3)
```

A. 7
B. Infinite Loop
C. 0
D. Error

Ans : C

5) What is the type of inf ?

a) Boolean

b) Integer

c) Float

d) Complex

Answer: c

Explanation: Infinity is a special case of floating point numbers. It can be obtained by float('inf'

6) Which of the following is incorrect?

a) x = 0b101
b) x = 0x4f5
c) x = 19023
d) x = 03964

Answer: d

Explanation: Numbers starting with a 0 are octal numbers but 9 isn't allowed in octal numbers.

7) What will be the output of the following Python code?

```
i = 1
while True:
        if i%0O7 == 0:
                break
        print(i)
        i += 1
```

a)1 2 3 4 5 6

b)1 2 3 4 5 6 7

c)Error

d) none of the mentioned

Answer: a

8) What will be the output of the following Python code?

```
for i in range(2.0):
    print(i)
```

a) 0.0 1.0
b) 0 1
c) error
d) none of the mentioned

Answer: c

Explanation: Object of type float cannot be interpreted as an integer.

9) What will be the output of the following Python code?

```
for i in range(int(2.0)):
    print(i)
```

a) 0.0 1.0
b) 0 1
c) error
d) none of the mentioned

Answer: b
Explanation: range(int(2.0)) is the same as range(2).

THANK YOU!