Module 1

Introduction to everything

Section

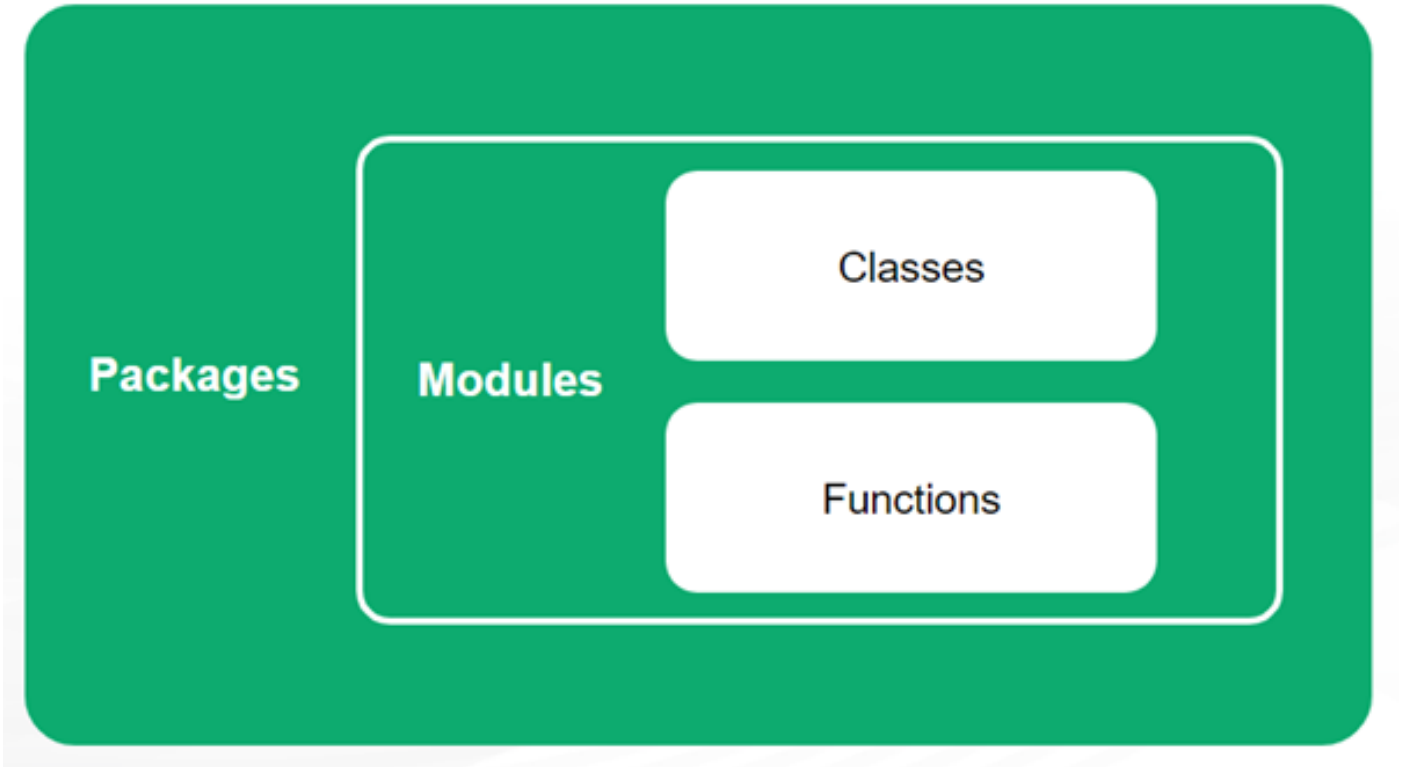
Python Programming: python data types & operators

Getting started with python

- Python is a highly efficient, interpreted, high-level, general purpose programming language.
- Python was created by Guido van Rossum and was first released in 1991 and is developed and maintained by the Python Software Foundation.
- The syntax of Python is simple and elegant and helps one write clean code. Python is so efficient that even a few lines of code will help achieving many tasks, compared to many other languages.
- Python code is easy to read, write debug, extend and build upon, compared to most other programming languages.

Components of python

The following picture illustrates the various components of Python and how they are organized-

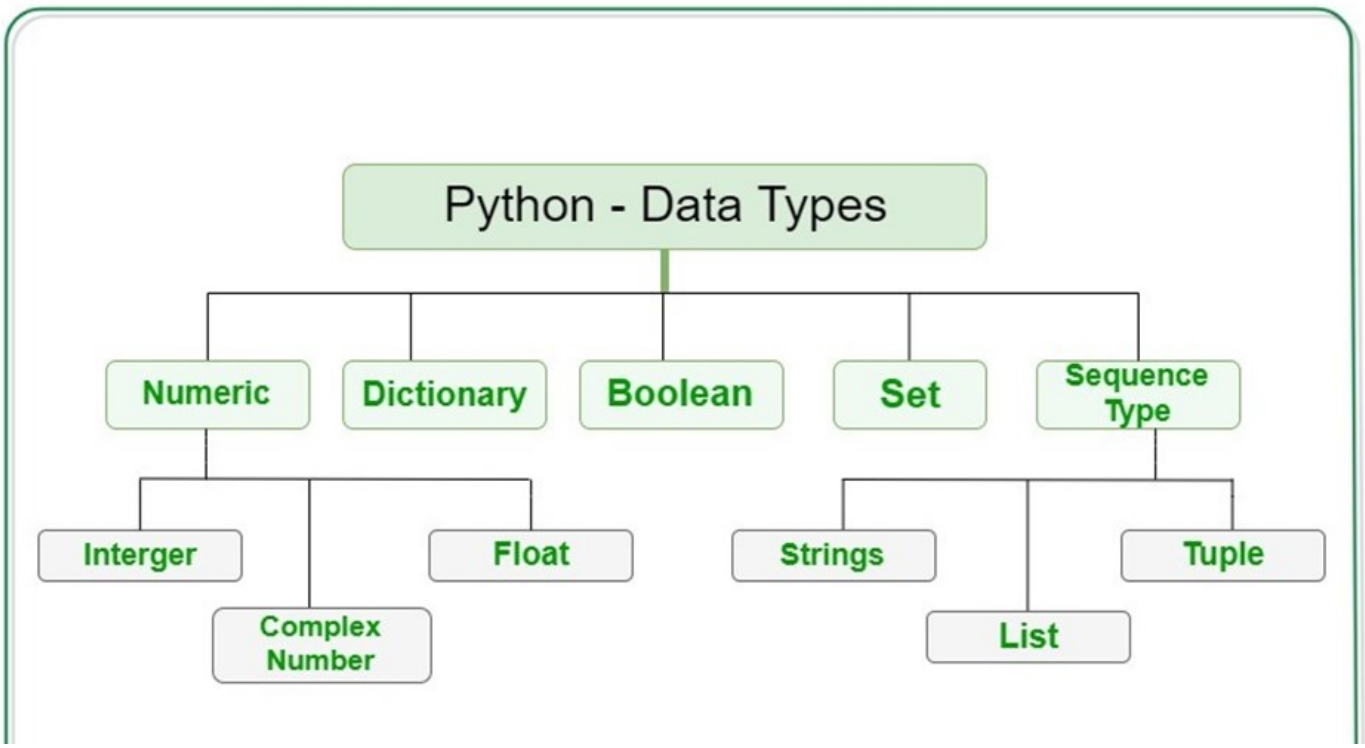


Python data types and operators

Python has a couple **base data types / primitive data types**:

- **int:** to represent integer numbers (0, 1, 2, 3, ...)
- **float:** to represent real numbers (0.1234, 1.2354, ...)
- **str:** to represent text ("Hello from Orbit!")
- **bool:** to represent things that can either be True or False (Booleans)
- **Complex:** to represent imaginary numbers (5+6j, -45j, 12-5j, ...)

Each data type has different behavior with different operators and functions. To understand what your code will do and to understand the error messages, it often boils down to understanding which data types are at work.

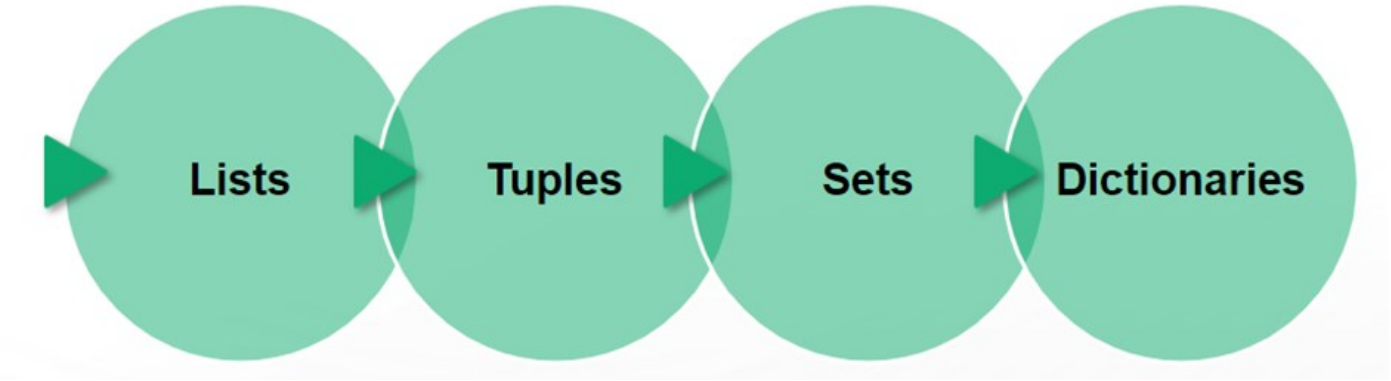


In Python, numeric datatype consists of –

- **int (integer)** - a whole number, positive or negative, without decimals, of unlimited length. Example: x = 5
- **float (floating point number)** - a number, positive or negative, containing one or more decimals. Example: y = 5.1
- **complex** - written with j as the imaginary part. Example: z = 5j

The **type ()** function is used to identify the type of the object.

Addition, subtraction, multiplication and division are the operations carried out on numbers.



Collections in python

LISTS

A list is a collection of items that is ordered and changeable. The key details of lists are as follows:

- A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging t

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list)           # Prints complete list
print(list[0])        # Prints first element of the list
print(list[1:3])      # Prints elements starting from 2nd till 3rd
print(list[2:])       # Prints elements starting from 3rd element
print(tinylist * 2)   # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

Output:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

TUPLES

A tuple is a collection of items that are ordered and unchangeable. The key details of tuples are as follows:

- Unchangeable values are referred to as ‘immutable’ in Python. A tuple is thus an immutable list.
- A tuple is similar to a list, but it is identified using a parenthesis, instead of a square bracket.

```
 tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
```

```
print(tuple)            # Prints complete tuple
print(tuple[0])         # Prints first element of the tuple
print(tuple[1:3])       # Prints elements starting from 2nd till 3rd
print(tuple[2:])        # Prints elements starting from 3rd element
print(tinytuple * 2)    # Prints tuple two times
print(tuple + tinytuple) # Prints concatenated tuple
```

Output:

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

DICTIONARY

A dictionary is a collection of key-value pairs, which are unordered, changeable and indexed. The key details of dictionaries are as follows:

- Python dictionaries are indicated by curly braces around the key-value pairs.
- Each key is connected to a value, and you can use a key to access the value associated with that key.
- A key's value can be a number, a string, a list, or even another dictionary.

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print(dict['one'])       # Prints value for 'one' key
print(dict[2])           # Prints value for 2 key
print(tinydict)          # Prints complete dictionary
print(tinydict.keys())   # Prints all the keys
print(tinydict.values()) # Prints all the values
```

Output:

```
This is one
This is two
{'name': 'john', 'code': 6734, 'dept': 'sales'}
```

```
dict_keys(['name', 'code', 'dept'])
dict_values(['john', 6734, 'sales'])
```

SETS

In Python, a set is a collection of items that are unordered and unindexed, but are mutable. The key details of sets are as follows:

- A set is indicated by means of curly braces. Being unordered, printing a set will output the items in a random order.
- Since sets are unordered, accessing items by means of indexing, is not possible, but like lists and tuples, looping through a

```
setA = {4, 5, 6, 3, 9}
setB = {1, 6, 2, 7, 5}

print(setA.union(setB))# Prints union of both the sets
print(setA.intersection(setB))# Prints intersection of both sets
```

Output:

```
{1, 2, 3, 4, 5, 6, 7, 9}
{5, 6}
```

OPERATORS IN PYTHON

1. **ARITHMETIC OPERATORS**

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

```
a,b,c=10,5,
(f"(Addition) {a}+{b} =",a+b)
(f"(Subtraction) {a}-{b} =",a)
(f"(Multiplication) {a}*{b} =",a)
(f"(Division) {a}/{b} =",a)
```

```
(f"(Exponentiation) {a}**{b} =",a)
(f"(Floor Division) {a}//{c} =",a)
```

Output:

```
(Addition) 10+5 = 15
(Subtraction) 10-5 = 5
(Multiplication) 10*5 = 50
(Division) 10/5 = 2.0
(Exponentiation) 10**5 = 100000
(Floor Division) 10//6 = 1
```

2. LOGICAL OPERATORS

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

```
x,y=4,3
print(x<5 and x<10)
print(x<5 or x<4)
print(not(x<5 and x<10))
```

Output:

```
 True
True
False
```

3. IDENTITY OPERATORS

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

```
indah=["car","bicycle"]
nur=["bicycle","airplane"]

print("Does Indah own the same vehicles as Nur?",x is y,"\n")
print("Is it true that Indah doesn't own the same vehicles as Nur?",x is not y)
```

Output:

```
Does Indah own the same vehicles as Nur? False

Is it true that Indah doesn't own the same vehicles as Nur? True
```

4. **MEMBERSHIP OPERATORS**

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

```
x=["Apples","Oranges",["Bananas","Mangoes"]]
y=["Bananas","Mangoes"]
print(y in x)
print(y not in x)
```

Output:

```
 True
False
```

5. **BITWISE OPERATORS**

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

```
a,b=10,4
print("Bitwise AND operation",a & b)
print("Bitwise OR operation",a | b)
print("Bitwise XOR operation",a^b)
print("Complement",~a)
print("Bitwise Right Shift",a >> 1)
print("Bitwise Left Shift",a << 1)
```

Output:

```
Bitwise AND operation 0
Bitwise OR operation 14
Bitwise XOR operation 14
Complement -11
Bitwise Right Shift 5
Bitwise Left Shift 20
```

6. **ASSIGNMENT OPERATORS**

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

```
x,y,z=2,5,10
print("Initial Value of x,y,z:",x,y,z)
x+=3
y-=3
z**=3
print("Final Value of x,y,z:", x,y,z)
```

Output:

```
Initial Value of x,y,z: 2 5 10
Final Value of x,y,z: 5 2 1000
```

Summary

- Python is a popular, easy to use, easy to update and maintain programming language.
- It is open-source, powerful, portable and has exhaustive set of libraries.
- Python is used today for variety of tasks, few popular applications include data analytics, machine learning, web scrapping, web application development, backend services development.
- Python 2.x is legacy, python 3.x is present and future of python
- Anaconda is a popular distribution of python with code editors such as Jupyter notebook and spyder IDE.
- Data structures are ways to store data in a programming language
- Python has five primitive data types which are immutable – integer, float, string, Boolean and complex
- Python has 4 non primitive data types – list, tuple, dictionary and set
- Tuple is immutable whereas list, dictionary and set are mutable.
- We can type cast primitive data types from one to another.