Datatypes in Python

What we will learn?

- Comments in python
- Docstrings
- Variables in python
- Built-in datatypes
- Sequences in python
- Sets datatype
- Mapping types
- Literals & constants in python
- Determining the datatype of a variable
- Identifiers & reserved words

Let us run our very first program in python

```
#Program to add two new numbers
a=5
b=7
c=a+b
print("Sum=",c)
```

```
Sum= 12
```

Comments in python

Comments are descriptions that help programmers better understand the intent and functionality of the program.

They are completely ignored by the Python interpreter.

It makes the program more readable which helps us remember why certain blocks of code were written. There are two types of comments in python:

1. Single line comments
2. Multi line comments

1. Single line comments:

   In Python, we use the hash symbol # to write a single-line comment.

   Example:

```
a=10 #store 10 into variable a
```

2. Multi line comments:

   There is no unique way to write multiline comments in Python. Python interpreter ignores the string literals that are not assigned to a variable. We can add a multiline string triple (single or double) quotes in code, and place comment inside it:

```
''' This is a multi line
comment '''
print('hello')
```

```
hello
```

```
""" This is a multi line
comment """
print('hello')
```

```
hello
```

Docstring

Python documentation strings (or docstrings) provide a convenient way of associating documentation with Python modules, functions, classes, and methods.

It's specified in source code that is used, like a comment, to document a specific segment of code. It should describe what the function does.

Declaring Docstrings: The docstrings are declared using ”’triple single quotes”’ or “””triple double quotes””” just below the class, method or function declaration. All functions should have a docstring.

Accessing Docstrings: The docstrings can be accessed using the __ doc __ method of the object or using the help function. The below examples demonstrates how to declare and access a docstring.

```
def square(n):
    '''Takes in a number n, returns the square of n'''
    return n**2
print("Using __doc__:")
print(square.__doc__)

print("Using help:")
help(square)
```

```
Using __doc__:
Takes in a number n, returns the square of n
Using help:
Help on function square in module __main__:

square(n)
    Takes in a number n, returns the square of n
```

Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable/identifier and the operand to the right of the = operator is the value stored in the variable. In Python, variables are a symbolic name that is a reference or pointer to an object. The variables are used to denote objects by that name.

Le's understand thiw with example:

```
a=50
```

In the above image, the variable a refers to an integer object.

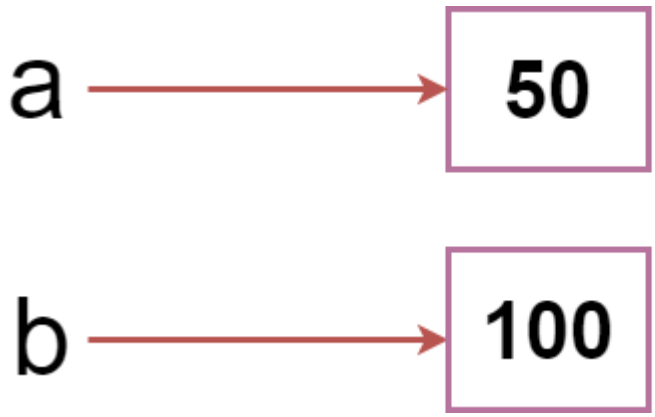
Suppose we assign the integer value 50 to a new variable b.

```
b = a
```

The variable b refers to the same object that a points to because Python does not create another object.

Let's assign the new value to b. Now both variables will refer to the different objects.

```
b=100
```



Assigning values:

```
a=1
b=1.1
msg='Python'
print(a)
print(b)
print(msg)
```

```
1
1.1
Python
```

Multiple assignments:

```
a,b,c=1,1.1,'Python'
print(c)
```

```
Python
```

Same value to multiple variables:

```
a=b=c=1
print(c)
```

```
1
```

Built-in datatypes

Data types are the classification or categorization of data items. Python supports the following built-in data types.

1. **None:**

   It represents the null object in Python. A None is returned by functions that don't explicitly return a value.

2. **Numeric:**

   Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

   int: It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.

   float: It is a real number with floating point representation. It is specified by a decimal point.

   complex: It is specified as (real part) + (imaginary part)j.

   For example – 2+3j

Converting datatype explicitly

1. float to int

```
a=10.57
print(int(a))
```

```
10
```

2. int to float

```
a=10
print(float(a))
```

```
10.0
```

3. int or float to complex

```
print(complex(a))
b=-15
print(complex(a,b))
```

```
(10+0j)
```

```
(10-15j)
```

type(): built-in function that prints type of object

```
a=1.0
b=2
c=complex(a+b)
print(type(a))
print(type(b))
print(type(c))
print(type(type(a)))
```

```
<class 'float'>
<class 'int'>
<class 'complex'>
<class 'type'>
```

3. **bool datatype:**

   Data type with one of the two built-in values, True or False. Python internally represents True as 1 and False as 0.

   A blank string "" is also represented as False.

   Example:

```
a=5>7
print(a)
```

```
False
```

```
print(True+True)
print(True-False)
a=(10>5)+(9>5)
print(a)
```

```
2
1
2
```

Sequences in python

It is a group of elements or items. It allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

1. **String datatype**

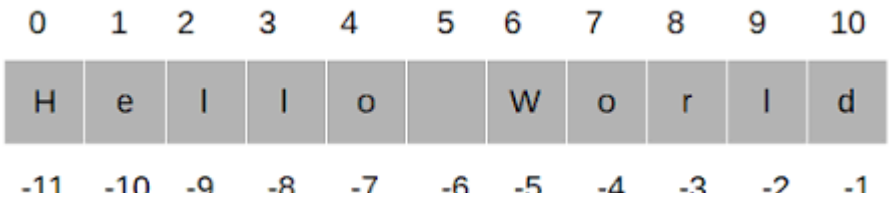
A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

```
String1 = 'Welcome to the python class'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm a Programmer"
print("\nString with the use of Double Quotes: ")
print(String1)
print(type(String1))

# Creating a String
# with triple Quotes
String1 = '''I'm learning Python'''
print("\nString with the use of Triple Quotes: ")
print(String1)
print(type(String1))

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Python
is
fun'''
print("\nCreating a multiline String: ")
print(String1)
```

```
String with the use of Single Quotes:
Welcome to the python class

String with the use of Double Quotes:
I'm a Programmer
<class 'str'>

String with the use of Triple Quotes:
I'm learning Python
<class 'str'>

Creating a multiline String:
Python
is
fun
```

Accessing string:

Individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String.




```
s1="Hello"
s2='hello'
s3='''This is 'core python' book'''
s4="welcome"
print(s1)
print(s2[1])
print(s3[5:7])
print(s4[3:])
print(s4[-1])
print(s1*2)
#to retrieve all characters of string we can use for loop
for i in s1:
  print(i)
```

```
Hello
e
is
come
e
HelloHello
H
e
l
l
o
```

2. **List datatype**

   Lists are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java).

   Lists need not be homogeneous always which makes it a most powerful tool in Python.

   A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are **mutable**, and hence, they can be altered even after their creation.

   It is created by placing all the items (elements) inside square brackets [], separated by commas.

```
list=[1,2.3,"abc",1+2j]
print(list)
print(list[0])
```

```
print(list[2:4])
print(list[-1])
print(list*2)
print(type(list))
```

```
[1, 2.3, 'abc', (1+2j)]
1
['abc', (1+2j)]
(1+2j)
[1, 2.3, 'abc', (1+2j), 1, 2.3, 'abc', (1+2j)]
<class 'list'>
```

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)   #ascending sort
thislist.sort(reverse = True) #descending sort
print(thislist)
```

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

3. **tuple datatype**

   Tuples like list are used to store multiple items in a single variable.

   A tuple is a collection which is ordered and unchangeable.

   Tuples are written with round brackets.

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print(tup3);
```

```
(12, 34.56, 'abc', 'xyz')
```

4. **range datatype**

   The range() type returns an immutable sequence of numbers between the given start integer to the stop integer.

   range() takes mainly three arguments having the same use in both definitions:

   start - integer starting from which the sequence of integers is to be returned

stop - integer before which the sequence of integers is to be returned.The range of integers ends at stop - 1.

step (Optional) - integer value which determines the increment between each integer in the sequence. Default value is 1.

Python does not unpack the result of the range() function. We can use argument-unpacking operator i.e. *.

```
My_list = [*range(10)]

# Print the list
print(My_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Create a list of even number between the given numbers using range()

```
My_list = [*range(2,14,2)]

print(My_list)
```

```
[2, 4, 6, 8, 10, 12]
```

Sets datatype

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

Sets are written with curly brackets.

```
s1={1,2,3,4}
print(s1)
s2=set("Hello")
print(s2)
l=[1,2,2,3]
s3=set(l)
print(s3)
s4=set("WOrlD")
print(s4)
```

```
{1, 2, 3, 4}
```

```
{'e', 'H', 'o', 'l'}
{1, 2, 3}
{'r', 'D', 'W', 'l', 'O'}
```

Indexing & slicng is not possible in sets as it is an unordered collection.

update() & remove() methods are use to add & delete elements respectively.

```
s3.update([5,6])
print(s3)
s3.remove(2)
print(s3)
```

```
{1, 2, 3, 5, 6}
{1, 3, 5, 6}
```

frozenset datatype

Frozen sets in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to which they are applied.

```
fs=frozenset(s3)
print(fs)
#fs.remove(3)
```

```
frozenset({1, 3, 5, 6})
```

Mapping types

A map represents a group of elements in the form of key value pairs so that when the key is given, we can retrieve the value associated with it.

Dictionary Data type

Python dictionary is an unordered collection of items. Each item of a dictionary has a key:value pair.

Dictionaries are optimized to retrieve values when the key is known.

Creating dictionary:

Creating a dictionary is as simple as placing items inside curly braces {} separated by commas.

An item has a key and a corresponding value that is expressed as a pair (key: value).

While the values can be of any data type and can repeat, keys must be of immutable type and must be unique.

```
d={}  #empty dictionary
print(d)
d={1:'MP',2:'AOA',3:'DBMS',4:'OS',5:'EM'}
print(d)
d[6]='PY' #d[key]=value
print(d)
```

```
{}
{1: 'MP', 2: 'AOA', 3: 'DBMS', 4: 'OS', 5: 'EM'}
{1: 'MP', 2: 'AOA', 3: 'DBMS', 4: 'OS', 5: 'EM', 6: 'PY'}
```

```
#Retrieve all keys
print(d.keys())
#Retrieve all values
print(d.values())
```

```
dict_keys([1, 2, 3, 4, 5, 6])
dict_values(['MP', 'AOA', 'DBMS', 'OS', 'EM', 'PY'])
```

```
#Retrieve value of given key
print(d[6])
#deleting key
del d[3]
print(d)
```

```
PY
{1: 'MP', 2: 'AOA', 4: 'OS', 5: 'EM', 6: 'PY'}
```

Literals in python

It is a constant value that is stored into a variale in a program. They can also be defined as raw value or data given in variables or constants.

The following are different types of literals in python:

1. Numeric literals

```
# Numeric literals
x = 24
y = 24.3
z = 2+3j
print(x, y, z)
```

```
24 24.3 (2+3j)
```

Here 24, 24.3, 2+3j are considered as literals.

2. Boolean literals

   There are only two boolean literals in Python. They are true and false.

2. String literals

   A string literal can be created by writing a text(a group of Characters) surrounded by the single(''), double(""), or triple quotes. By using triple quotes we can write multi-line strings or display in the desired way.

```
# in single quote
s = 'Python'

# in double quotes
t = "Programmin"

# multi-line String
m = '''Python
           Programming'''

print(s)
print(t)
print(m)
```

```
     Python
     Programmin
     Python
              Programming
```

Determining the datatype of a variable

type(): built-in function that prints type of object.

```
ch='A'
print(type(ch))
```

```
     <class 'str'>
```

Python identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another. Rules for writing identifiers:

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _.
2. Names like myClass, var_1 and print_this_to_screen, all are valid example.
3. An identifier cannot start with a digit. 1variable is invalid, but variable1 is a valid name.
4. Keywords cannot be used as identifiers. eg: for=1 is invalid
5. We cannot use special symbols like !, @, #, $, % etc. in our identifier.
6. An identifier can be of any length.

Python keywords

Keywords are the reserved words in Python.

We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.All the keywords except True, False and None are in lowercase and they must be written as they are. The list of some keywords is 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	