02 Variables and Data Types

Variables

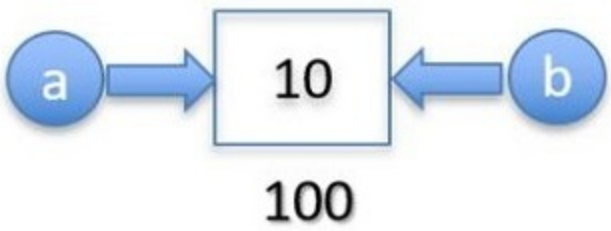
Variables are the memory locations used to store values with in a Program.

Data items belonging to different data types are stored in computer's memory. Computer's memory locations are having a number or address, internally represented in binary form. built-in **id()** function returns the address of an object/variable.

```
month="May"
age=18

print(id(month)) # >>> 2167264641264
print(id(age))   # >>> 140714055169352
```

A Python variable refers to the object and not the memory location. An object is stored in memory only once. Multiple variables are really the multiple labels to the same object.




```
a = 10
b = 10
print(a is b)    # >>> True
print(id(a), id(b))    # >>> 140731955278920, 140731955278920
```

Python's identity operator **is** returns True if both the operands have same id() value.

Creating Variables
Python variables do not need explicit declaration to reserve memory space or you can say to create a variable. A Python variable is created automatically when you assign a value to it. The equal sign (=) is used to assign values to variables.

```
counter = 100 # Creates an integer variable
miles = 1000.0 # Creates a floating point variable
```

```
name = "Student Name" # Creates a string variable
```

Deleting Python Variables

You can delete the reference to a variable/object by using the del statement.

```
del var_a, var_b
```

Type of a Variable

You can get the data type of a Python variable using the python built-in function type()

```
x = "Zara"
y = 10
z = 10.10

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

Variables - Multiple Assignment

Python allows to initialize more than one variables in a single statement.

```
a=b=c=10
print (a,b,c)    # >>> 10 10 10

a,b,c = 10,20,"Student"
print (a,b,c)    # >>> 10 20 Student
```

Local Variables

Python Local Variables are defined inside a block. We can not access variable outside the block.

```
def sum(x,y):    # x, y are local to sum function
        sum = x + y
        return sum
print(sum(5, 10))
```

Global Variables

Any variable created outside a block can be accessed within any block and so they have global scope.

```
x = 5    # global variable
y = 10    # global variable
def sum():
        sum = x + y
        return sum
print(sum())
```

Constants
Python doesn't have any formally defined constants. You can follow naming convention to indicate constants by using all-caps names with underscores.

```
ZIP_CODE = 500001
```

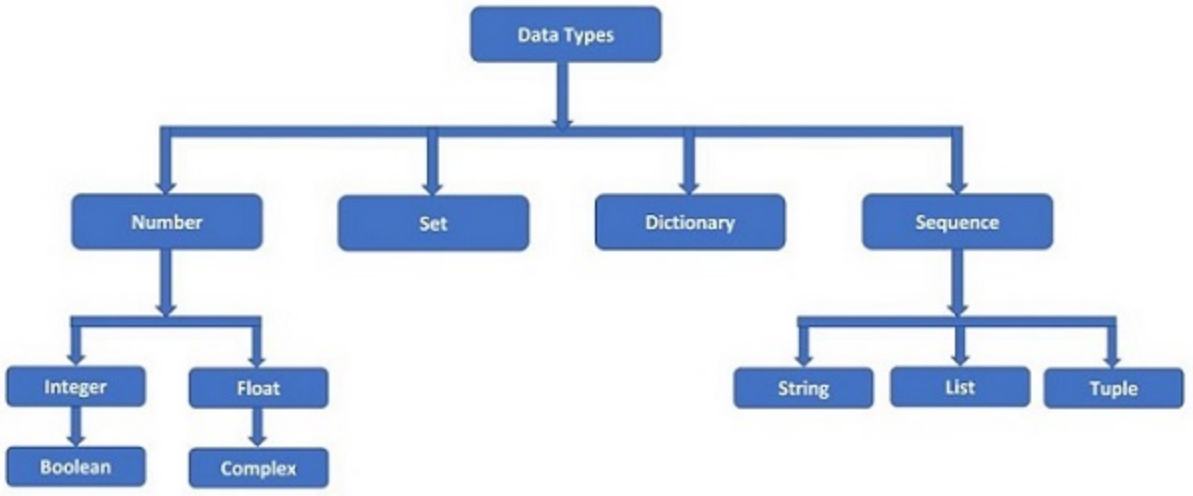
Data Types

Python data types are actually classes, and the defined variables are their instances or objects. Since Python is dynamically typed, the data type of a variable is determined at runtime based on the assigned value.

Types of Data Types

- Numeric Data Types
  - int
  - float
  - complex
- String Data Types
- Sequence Data Types
  - list
  - tuple
  - range
- Binary Data Types
  - bytes
  - bytearray
  - memoryview
- Dictionary Data Type
- Set Data Type
  - set
  - frozenset
- Boolean Data Type

- None Type



Numeric Data Types

Numeric data types store numeric values. Number objects are created when you assign a value to them.

```
var1 = 1 # int data type
var2 = True # bool data type
var3 = 10.023 # float data type
var4 = 10+3j # complex data type
```

A complex number is made up of two parts - **real** and **imaginary**. They are separated by '+' or '-' signs. The imaginary part is suffixed by 'j' which is the imaginary number.

```
print(type(5+6j))    # >>> <class 'complex'>
```

String Data Type

String is a sequence of one or more Unicode characters, enclosed in single, double or triple quotation marks (also called inverted commas). Python strings are immutable which means when you perform an operation on strings, you always produce a new string object of the same type, rather than mutating an existing string.

```
str = 'Hello World!'
type(str)    # >>> <class 'str'>
print (str[2:])    # >>> llo World!
print (str + "TEST")    # >>> Hello World!TEST
```

Sequence Data Types

Sequence is a collection data type. It is an ordered collection of items. Items in the sequence have a positional index starting with 0.

List Data Type

A list contains items separated by commas and enclosed within square brackets ([]).

```
print(type([2023, "Python", 3.11, 5+6j, 1.23E-4]))    # >>> <class 'list'>
```

Tuple Data Type

A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses (...).

```
print(type((2023, "Python", 3.11, 5+6j, 1.23E-4)))    # >>> <class 'tuple'>
```

The main differences between lists and tuples are: lists are mutable, while tuples are immutable.

Range Data Type

A Python range is an immutable sequence of numbers which is typically used to iterate through a specific number of items. It is represented by the **Range** class. The constructor of this class accepts a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number.

`range(start, stop, step)`

- **start**: Integer number to specify starting position, (Its optional, Default: 0)
- **stop**: Integer number to specify ending position (It's mandatory)
- **step**: Integer number to specify increment, (Its optional, Default: 1)

```
for i in range(5):
        print(i)
# >>> 0
# >>> 1
# >>> 2
# >>> 3
# >>> 4
```

Binary Data Types

The byte data type represents a sequence of bytes. We can create bytes using the built-in **bytes()** function or by prefixing a sequence of numbers with **b**.

```
b1 = bytes([65, 66, 67, 68, 69])
print(b1)    # >>> b'ABCDE'
```

Using the "b" prefix before a string to automatically create a bytes object.

```
b2 = b'Hello'
print(b2)    # >>> b'Hello'
```

Bytearray Data Type

The bytearray data type in Python is quite similar to the bytes data type, but with one key difference: it is mutable, meaning you can modify the values stored in it after it is created.

```
# Creating a bytearray from an iterable of integers
value = bytearray([72, 101, 108, 108, 111]) print(value)    # >>>
bytearray(b'Hello')
```

Creating a bytearray by encoding a string using a "UTF-8" encoding

```
# Creating a bytearray by encoding a string
val = bytearray("Hello", 'utf-8')
print(val)    # >>>> bytearray(b'Hello')
```

Memoryview Data Type

A memoryview is a built-in object that provides a view into the memory of the original object, generally objects that support the buffer protocol, such as byte arrays (bytearray) and bytes (bytes). It allows you to access the underlying data of the original object without copying it, providing efficient memory access for large datasets.

```
data = bytearray(b'Hello, world!')
view = memoryview(data)
print(view)    # >>>> <memory at 0x00000186FFAA3580>
```

If you have an array object, you can create a memoryview using the buffer interface.

```
import array
arr = array.array('i', [1, 2, 3, 4, 5])
view = memoryview(arr)
print(view)    # >>> <memory at 0x0000017963CD3580>
```

You can also create a memoryview by slicing a bytes or bytearray object.

```
data = b'Hello, world!'
# Creating a view of the last part of the data
```

```
view = memoryview(data[7:])
print(view)    # <memory at 0x00000200D9AA3580>
```

Dictionary Data Type

Dictionaries are kind of hash table type. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

```
d = {1:'one', 2:'two', 3:'three'}
print(type(d))    # >>> <class 'dict'>
d[2] = "This is two"
print(dict[2])    # >>> This is two
```

Set Data Type

Set is a Python implementation of set as defined in Mathematics. A set in Python is a collection, but is not an indexed or ordered collection as string, list or tuple.

```
print(type({2023, "Python", 3.11, 5+6j, 1.23E-4})) # >>> <class 'set'>
```

A set can store only **immutable** objects such as number (int, float, complex or bool), string or tuple. If you try to put a list or a dictionary in the set collection, Python raises a **TypeError**.

Boolean Data Type

The **boolean** type is one of built-in data types which represents one of the two values either **True** or **False**. Python **bool()** function allows you to evaluate the value of any expression and returns either True or False based on the expression.

```
a = True
print(type(a))    # >>> <class 'bool'>

c = 2 d = 4
print(bool(c==d))    # >>> False
```

None Type

None type is represented by the "nonetype." It is an object of its own data type.
The nonetype represents the null type of values or absence of a value.

```
x = None
print("type of x = ", type(x))    # >>> <class 'NoneType'>
```

Primitive and Non-primitive Data Types

Primitive Types: The primitive data types are the fundamental data types that are used to

create complex data types. Ex: Integers, Floats, Booleans and Strings

Non-primitive Types: The non-primitive data types store values or collections of values. Ex: Lists, Tuples, Dictionaries and Sets