Python Datatypes

- Everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

- Standard Built-in types
  - **Numeric**
  - **Sequence Type**
  - **Boolean**
  - **Set**
  - **Dictionary**

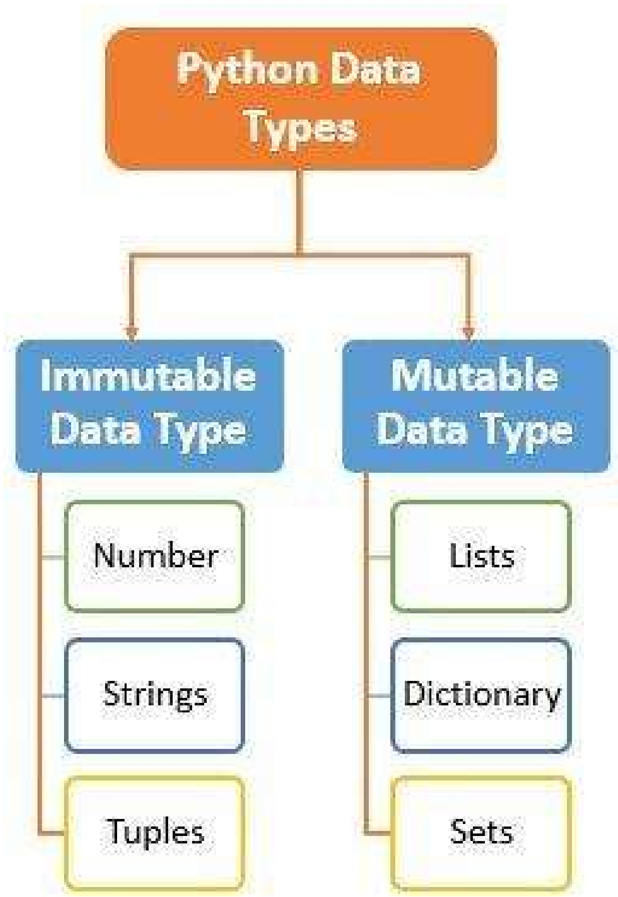
Python Datatypes

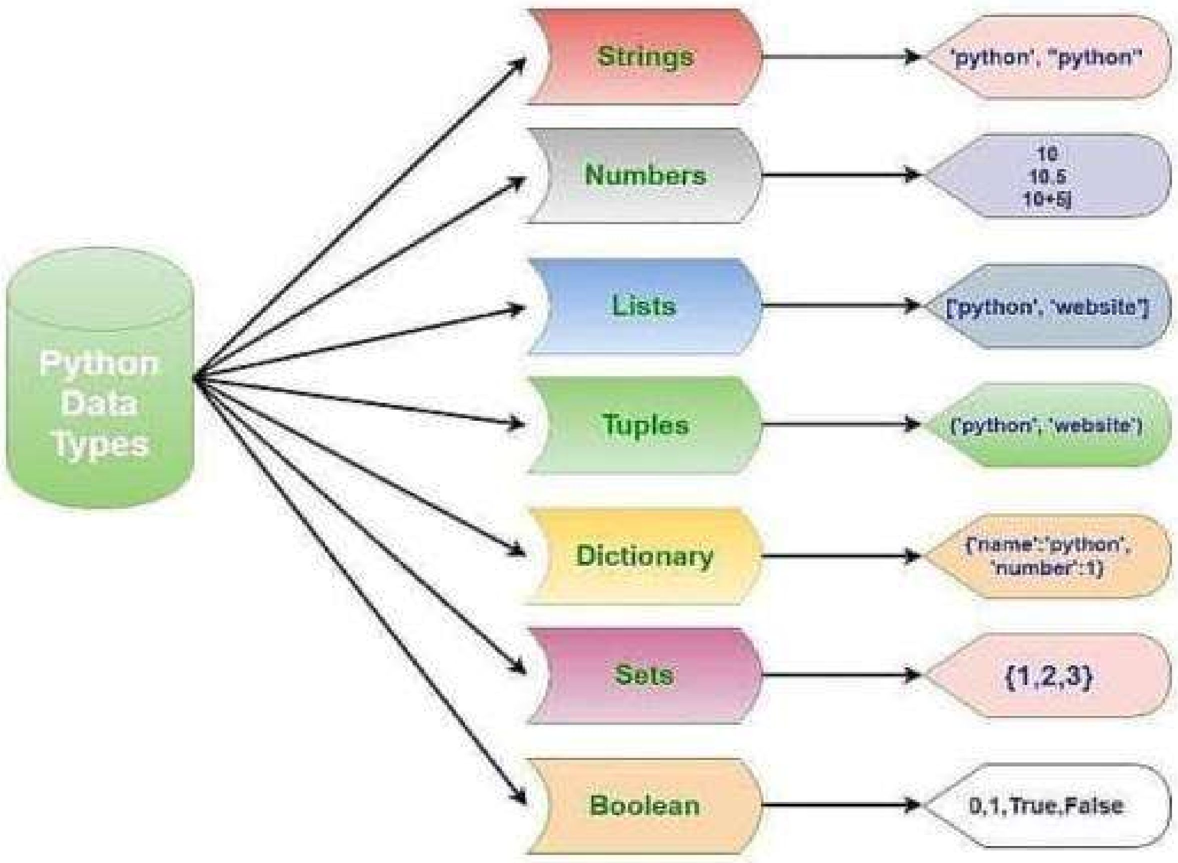
Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Python Datatypes

Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

Python Datatypes

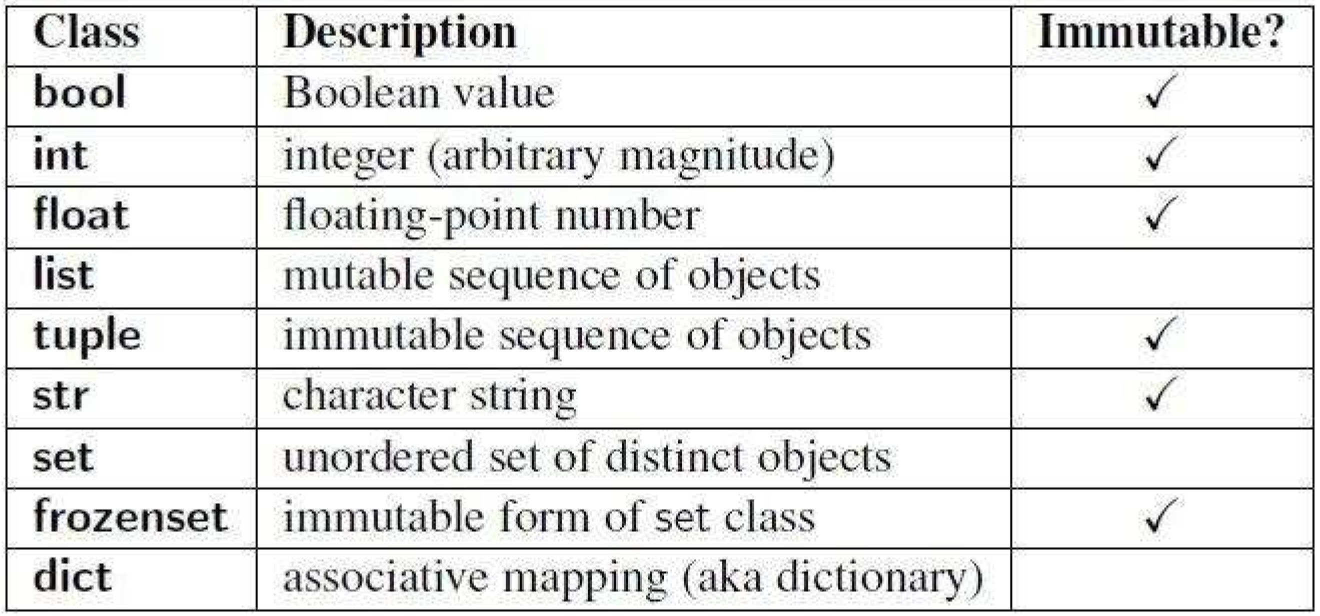




Python Datatypes

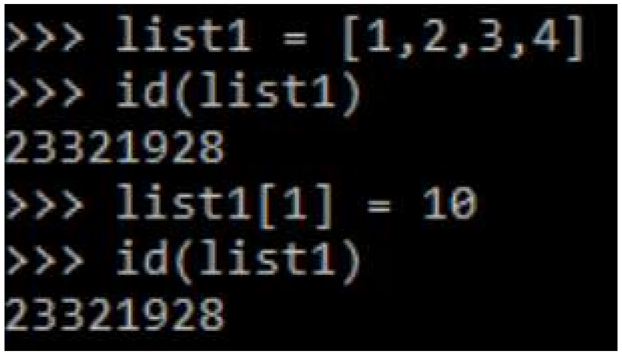
- A **mutable** object can be changed after it is created, and an **immutable** object can't.

- mutable objects by changing the element the id() value will not change , whereas for immutable objects *id()* value will change.



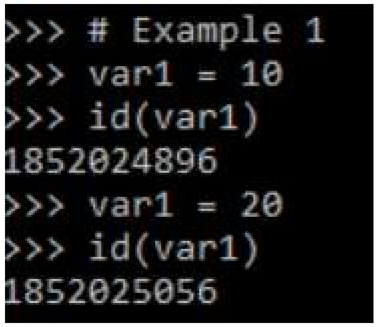
Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Mutable objects:
list, dict, set, byte array

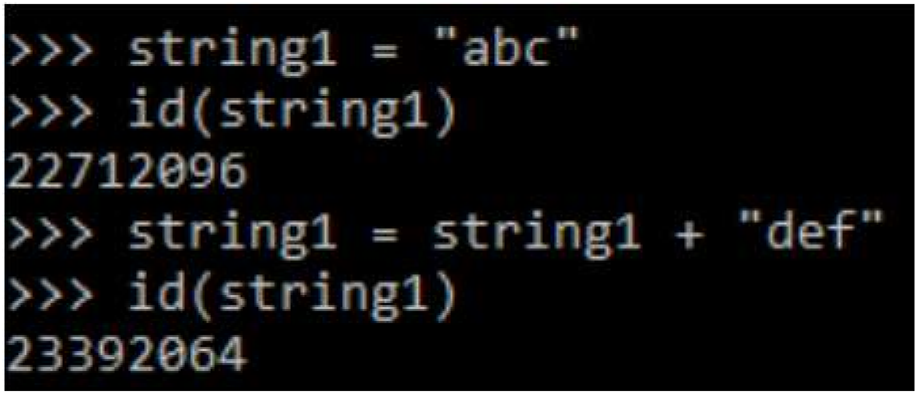


```
>>> list1 = [1,2,3,4]
>>> id(list1)
23321928
>>> list1[1] = 10
>>> id(list1)
23321928
```

Immutable objects:
int, float, complex, string, tuple, frozen set
[note: immutable version of set], bytes



```
>>> # Example 1
>>> var1 = 10
>>> id(var1)
1852024896
>>> var1 = 20
>>> id(var1)
1852025056
```



```
>>> string1 = "abc"
>>> id(string1)
22712096
>>> string1 = string1 + "def"
>>> id(string1)
23392064
```

Declaring and using Numeric data types

- Integer
- Float
- Complex
- String

Integer

- Integers are one of the Python data types. An integer is a whole number, negative, positive or zero.
- In Python, integer variables are defined by assigning a whole number to a variable. Python's type() function can be used to determine the data type of a variable.

```
>>> a = 5
>>> type(a)
<class 'int'>
```

Float

- Floating point numbers or floats are another Python data type.
- Floats are decimals, positive, negative and zero.
- Floats can also be represented by numbers in scientific notation which contain exponents.
- Both a lower case e or an upper case E can be used to define floats in scientific notation.
- In Python, a float can be defined using a decimal point . when a variable is assigned.

```
>>> c = 6.2
>>> type(c)
<class 'float'>
>>> d = -0.03
>>> type(d)
<class 'float'>
>>> Na = 6.02e23
>>> Na
6.02e+23
>>> type(Na)
<class 'float'>
```

Complex

- Another useful numeric data type for problem solvers is the complex number data type.
- A complex number is defined in Python using a real component + an imaginary component j.
- The letter j must be used to denote the imaginary component.
- Using the letter i to define a complex number returns an error in Python.

```
>>> comp = 4 + 2j
>>> type(comp)
<class 'complex'>
```

String

- Numbers and decimals can be defined as strings too. If a decimal number is defined using quotes ' ', the number is saved as a string rather than as a float.
- Integers defined using quotes become strings as well.

```
>>> num = '5.2'
>>> type(num)
<class 'str'>

>>> num = '2'
>>> type(num)
<class 'str'>
```

Thank You