S4 , S5

Programming in


A presentation by:

- Arun Kumar

- Variable
- Variable data Assignment
- Multiple variable data assignment
- Data types: Standard and derived
- Data Type Conversion

# Variable

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.

Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

**Example :**

```
Name = "Arun"
Height = 182
Weight = 70.5

print(Name)
print(Height)
```

# Assigning Values to 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 and the operand to the right of the = operator is the value stored in the variable.

For example −

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
Name    = "Arun"       # A string

print(counter)
print(miles)
print(Name)
```

Here, 100, 1000.0 and “Arun" are the values assigned to *counter*, *miles*, and *Name* variables, respectively.

This produces the following result −

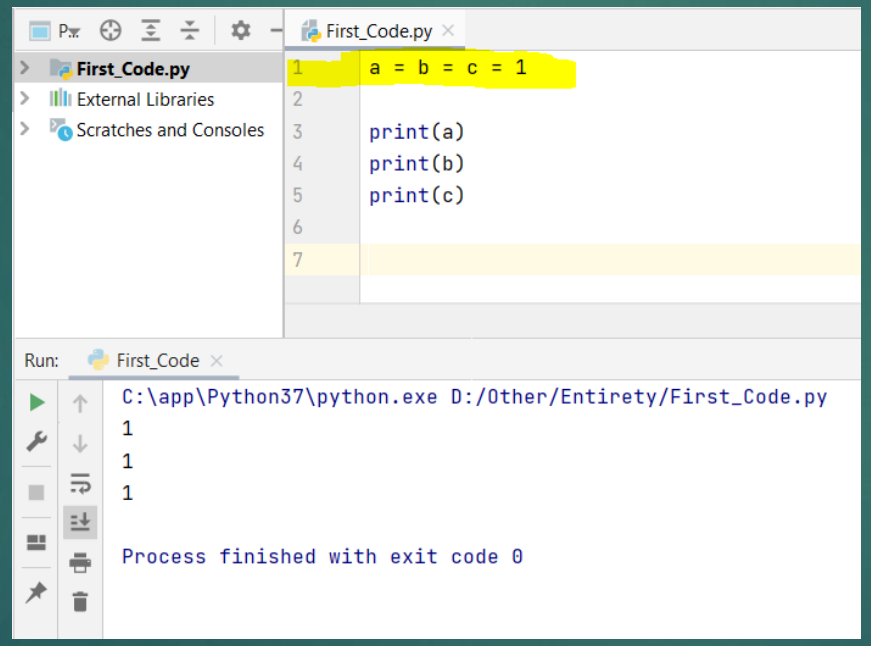
100
1000.0
Arun

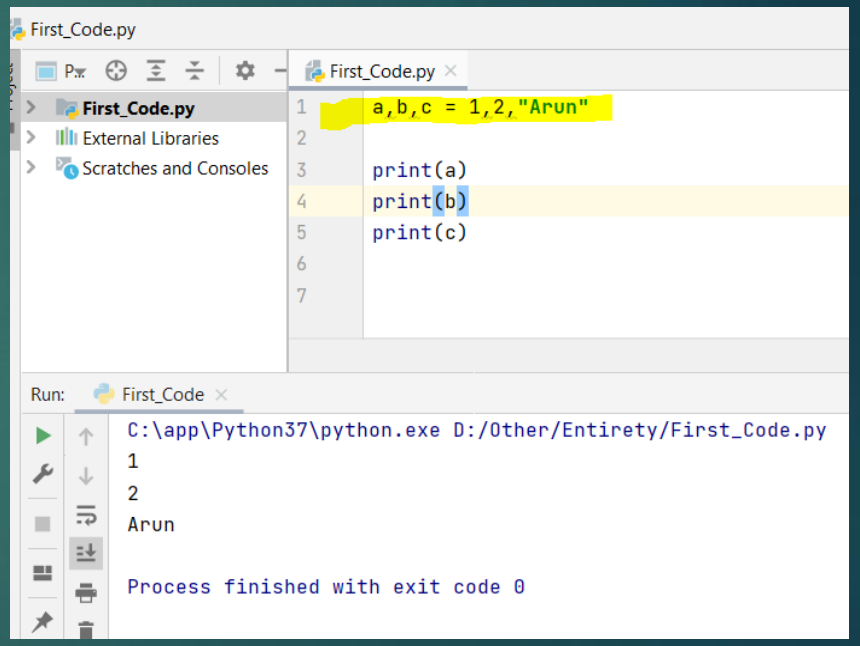
# Multiple Assignment

Python allows you to assign a single value to several variables simultaneously.

For example –

Two different ways of multiple assignments :



# Data-Types in Python

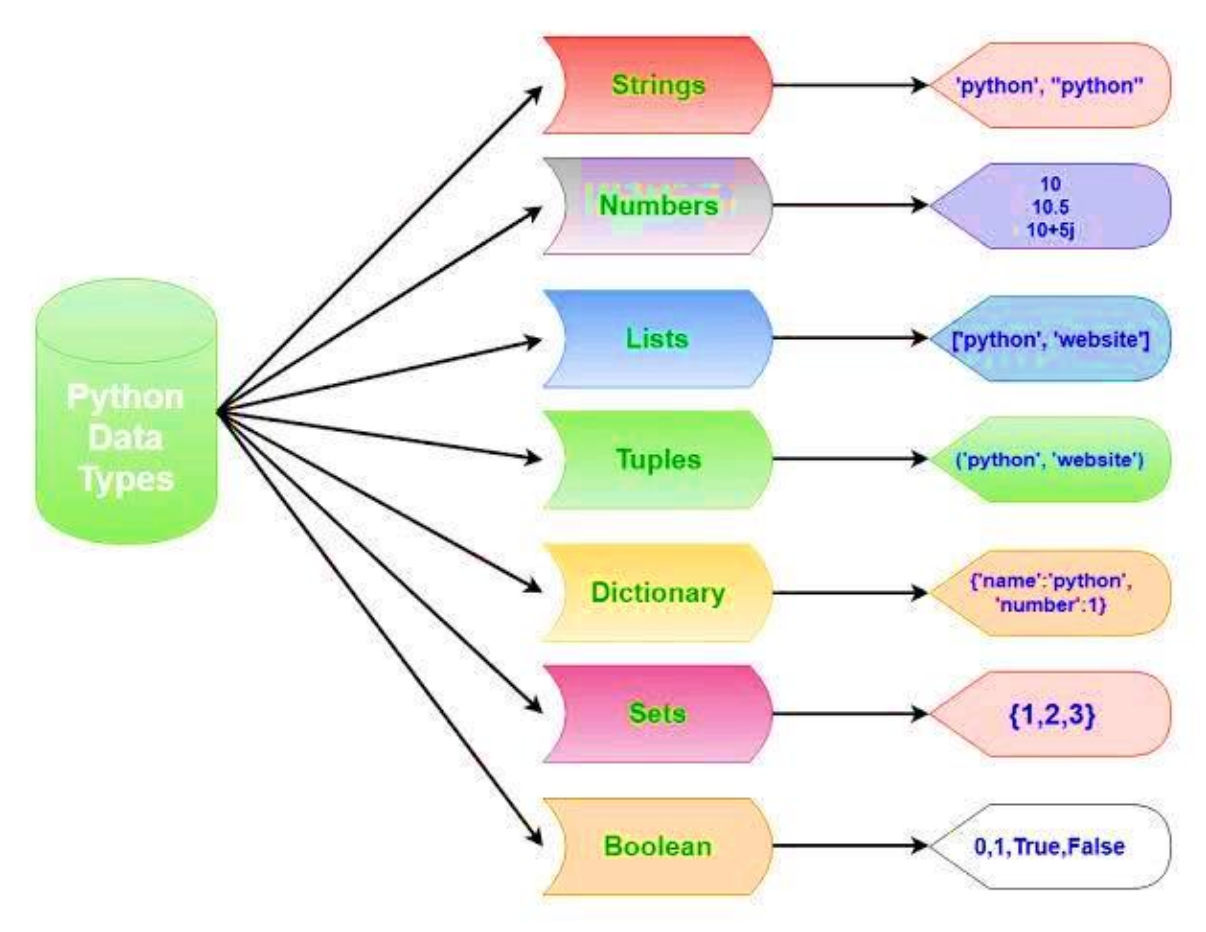
Standard Data Types

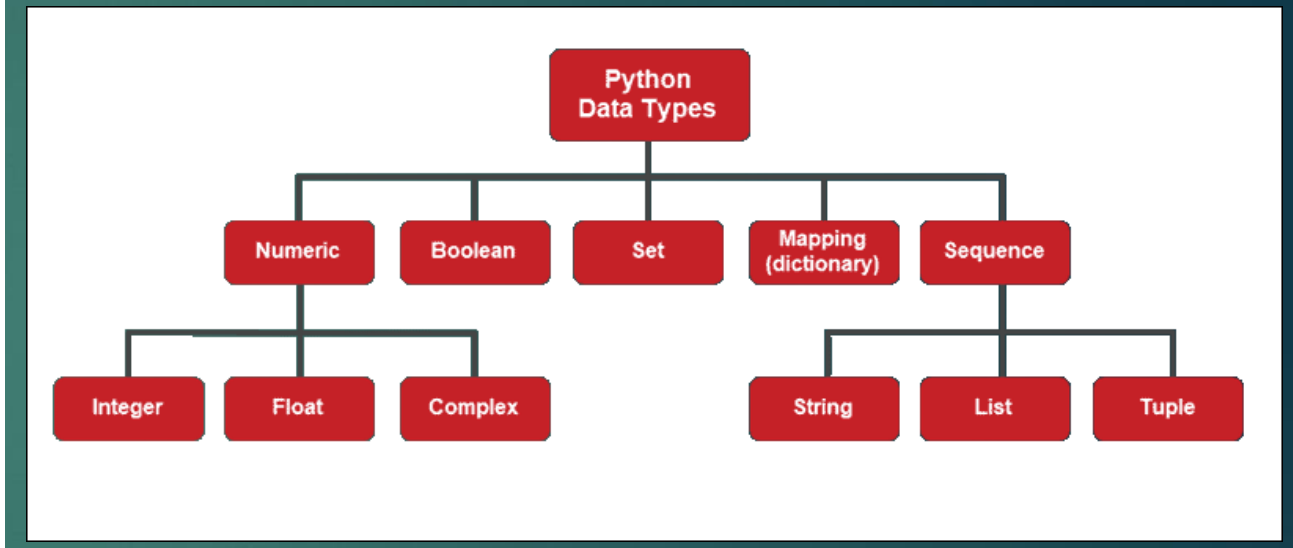
The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types −

- Numbers
- String
- List
- Tuple
- Dictionary

# DataTypes in Python



Data Type examples :

| Name | Data Type | Description |
|---|---|---|
| **Integer** | int | a number that can be written without fractions: 22 10 0 -300 |
| **Boolean** | bool | logical value that indicates **True** or **False** |
| **Floating-point** | float | a number that has a decimal component: 3.14 2.73 10.0 |
| **String** | str | sequence of characters: "hello world" "hey88340" '2018' |
| **List** | list | sequence of comma-separated numbers, strings etc.: [10, '2018', "hi"] |
| **Dictionary** | dict | collection of key-value pairs: {"key1": "value1", "key2": "value2"} |
| **Tuple** | tuple | sequence of comma-separated numbers, strings etc.: (10, 20.0, "world", 5) |

# DataType : Numbers

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

For example −

```
var1 = 1
var2 = 10
```

You can delete a single object or multiple objects by using the del statement.

For example −

```
del var1
del var_2
```

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

| int | long | float | complex |
|---|---|---|---|
| 10 | 51924361L | 0.0 | 3.14j |
| 100 | -0x19323L | 15.20 | 45.j |
| -786 | 0122L | -21.9 | 9.322e-36j |
| 080 | 0xDEFABCECBDAECBFBAEl | 32.3+e18 | .876j |
| -0490 | 535633629843L | -90. | -.6545+0J |
| -0x260 | -052318172735L | -32.54e100 | 3e+26J |
| 0x69 | -4721885298529L | 70.2-E12 | 4.53e-7j |

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.

- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

# DataType : Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

For example –

First_Code.py

```
ActorName = 'Mohanlal Viswanathan'

print(ActorName)                # Prints complete string
print(ActorName[0])             # Prints first character of the string
print(ActorName[2:5])           # Prints characters starting from 3rd to 5th
print(ActorName[2:])            # Prints string starting from 3rd character
print(ActorName * 2)            # Prints string two times
print(ActorName + "Mathews")    # Prints concatenated string
```

Run: First_Code

```
C:\app\Python37\python.exe D:/Other/Entirety/First_Code.py
Mohanlal Viswanathan
M
han
hanlal Viswanathan
Mohanlal ViswanathanMohanlal Viswanathan
Mohanlal ViswanathanMathews

Process finished with exit code 0
```

# DataType : Lists

- A list contains items separated by commas and enclosed within square brackets [ ].
- The values stored in a list can be accessed using the slice operator ( [ ] and [:] ) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator

For example −

```python

Mylist = [ 'India', 786 , 2.23, 'Mohan', 70.2 ]
tinylist = [123, 'Mohan']

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

```
C:\app\Python37\python.exe D:/Other/Entirety/First_Code.py
['India', 786, 2.23, 'Mohan', 70.2]
India
[786, 2.23]
[2.23, 'Mohan', 70.2]
[123, 'Mohan', 123, 'Mohan']
['India', 786, 2.23, 'Mohan', 70.2, 123, 'Mohan']
```

# DataType : Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated. Tuples can be thought of as **read-only** lists.

For example −

First_Code.py
External Libraries
Scratches and Consoles

```python
Mytuple = ( 'India', 786 , 2.23, 'Mohan', 70.2  )
tinytuple = (123, 'Mohan')

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

Run: First_Code

```
C:\app\Python37\python.exe D:/Other/Entirety/First_Code.py
('India', 786, 2.23, 'Mohan', 70.2)
India
(786, 2.23)
(2.23, 'Mohan', 70.2)
(123, 'Mohan', 123, 'Mohan')
('India', 786, 2.23, 'Mohan', 70.2, 123, 'Mohan')

Process finished with exit code 0
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. However, Similar case is possible with lists −

```
Mytuple = ( 'India', 786 , 2.23, 'Mohan', 70.2  )
Mylist = [ 'India', 786 , 2.23, 'Mohan', 70.2  ]
Mytuple[2] = 1000     # Invalid syntax with tuple
Mylist[2] = 1000      # Valid syntax with list
```

```
C:\app\Python37\python.exe D:/Other/Entirety/First_Code.py
Traceback (most recent call last):
  File "D:/Other/Entirety/First_Code.py", line 3, in <module>
    Mytuple[2] = 1000    # Invalid syntax with tuple
TypeError: 'tuple' object does not support item assignment

Process finished with exit code 1
```

## DataType : Dictionary

- Python's dictionaries are kind of hash table type.
- They work like associative arrays or hashes found in Perl and consist of key-value pairs. 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.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

For example –

```
First_Code.py
External Libraries
Scratches and Consoles

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

tinydict = {'name': 'Arun','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
```

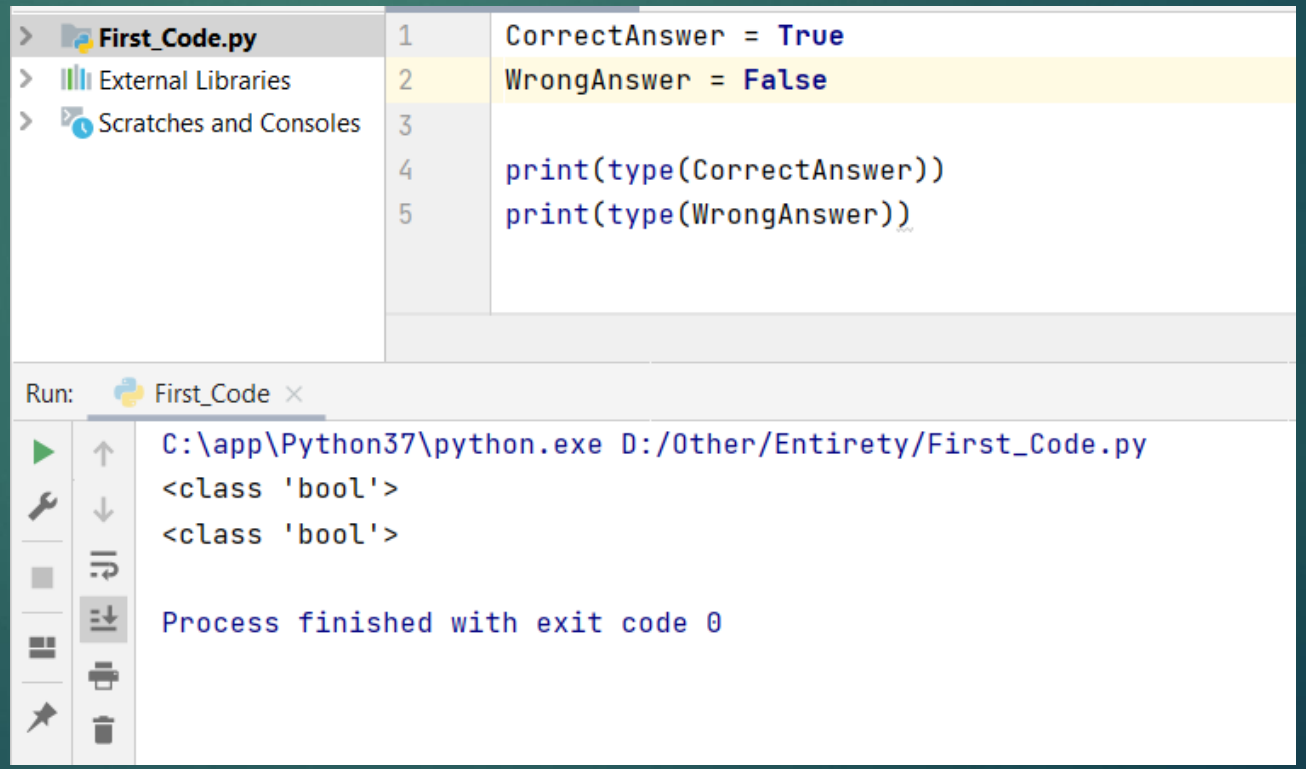
```
Run: First_Code

C:\app\Python37\python.exe D:/Other/Entirety/First_Code.py
This is one
This is two
{'name': 'Arun', 'code': 6734, 'dept': 'sales'}
dict_keys(['name', 'code', 'dept'])
dict_values(['Arun', 6734, 'sales'])

Process finished with exit code 0
```

## DataType : Boolean

- **boolean** type is one of the built-in data types provided by Python, which represents one of the two values i.e. True or False.
  Generally, it is used to represent the truth values of the expressions.

- For example, 1== 0 is True whereas 2<1 is False

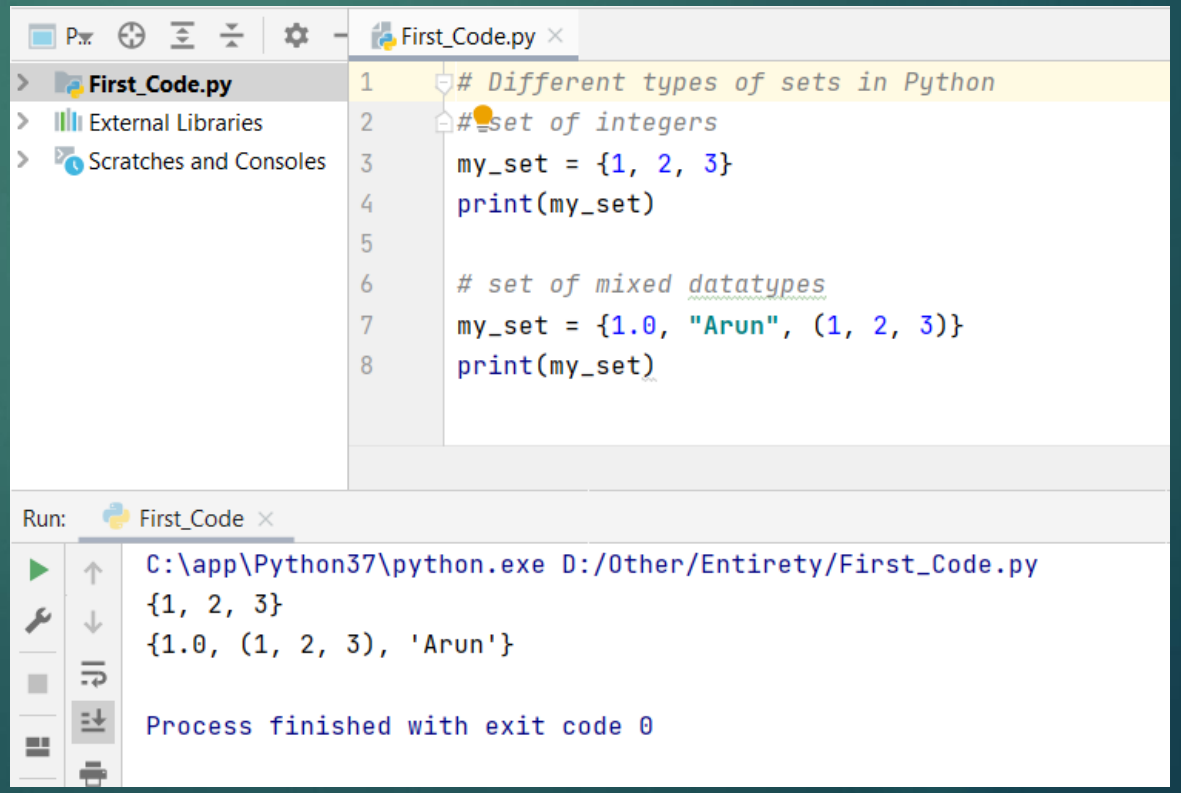
For example –



# DataType : Sets

- A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).However, a set itself is mutable. We can add or remove items from it.
- Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.
- A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

For example –



# Data Type conversions

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

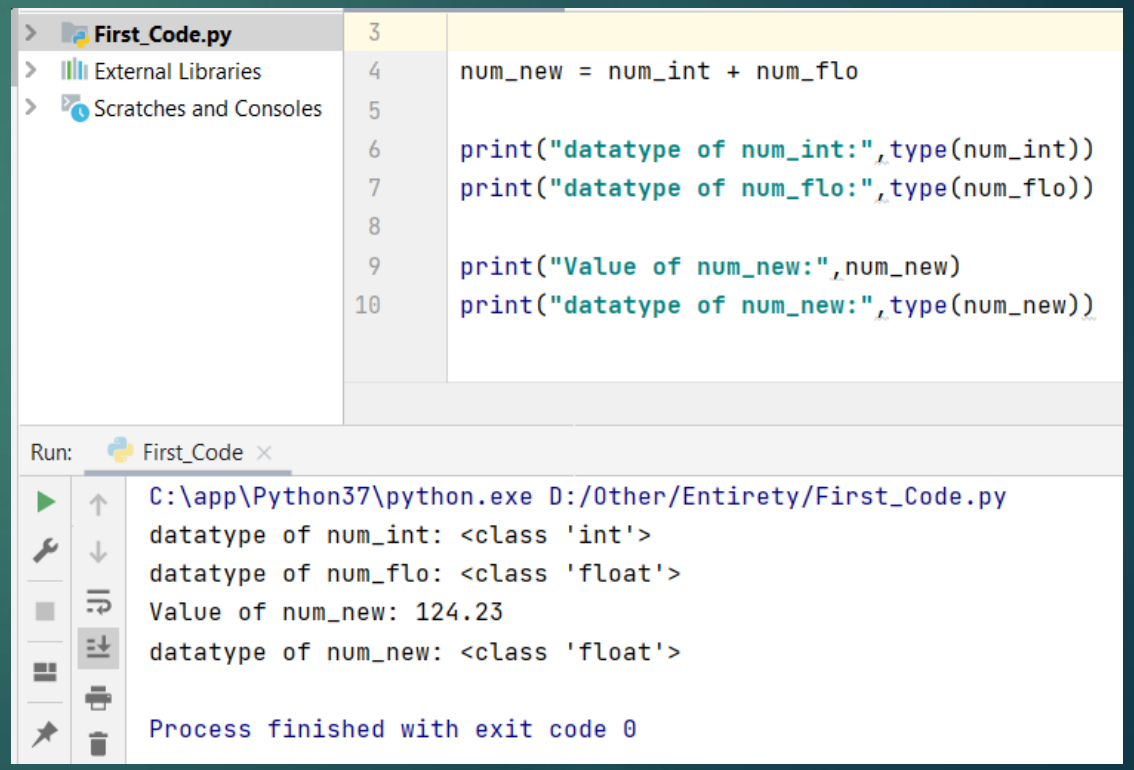
1.Implicit Type Conversion

2.Explicit Type Conversion

## Implicit Type Conversion

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Eg: where Python promotes the conversion of the lower data type (integer) to the higher data type (float) to avoid data loss.



## Explicit Type Conversion

In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

First_Code.py

```python
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:",type(num_str))

num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

num_sum = num_int + num_str

print("Sum of num_int and num_str:",num_sum)
print("Data type of the sum:",type(num_sum))
```

Run: First_Code

```
C:\app\Python37\python.exe D:/Other/Entirety/First_Code.py
Data type of num_int: <class 'int'>
Data type of num_str before Type Casting: <class 'str'>
Data type of num_str after Type Casting: <class 'int'>
Sum of num_int and num_str: 579
Data type of the sum: <class 'int'>

Process finished with exit code 0
```