

Python Data Types

A Data Type describes the characteristics of a variable.

Python has six standard Data Types:

- Numbers
- String
- List
- Tuple
- Set
- Dictionary

#1) Numbers

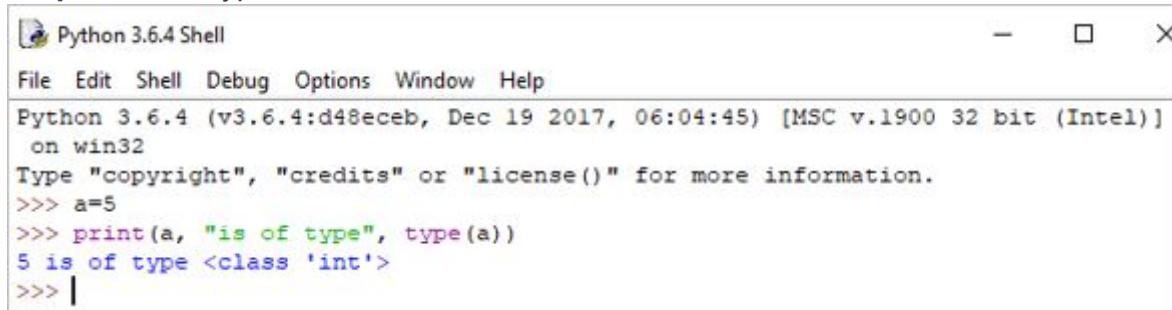
In Numbers, there are mainly 3 types which include **Integer**, **Float**, and **Complex**.

These 3 are defined as a class in python. In order to find to which class the variable belongs to you can use **type()** function.

Example:

```
1 a = 5
2 print(a, "is of type", type(a))
```

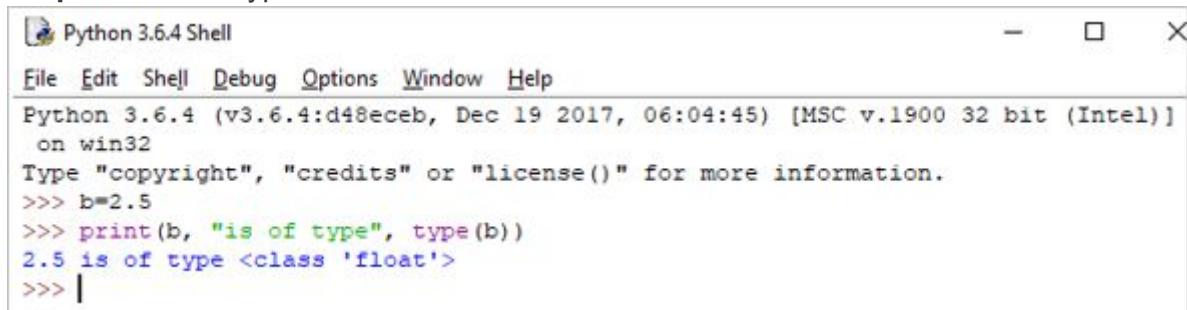
Output: 5 is of type <class 'int'>



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=5
>>> print(a, "is of type", type(a))
5 is of type <class 'int'>
>>> |
```

```
1 b = 2.5
2 print(b, "is of type", type(b))
```

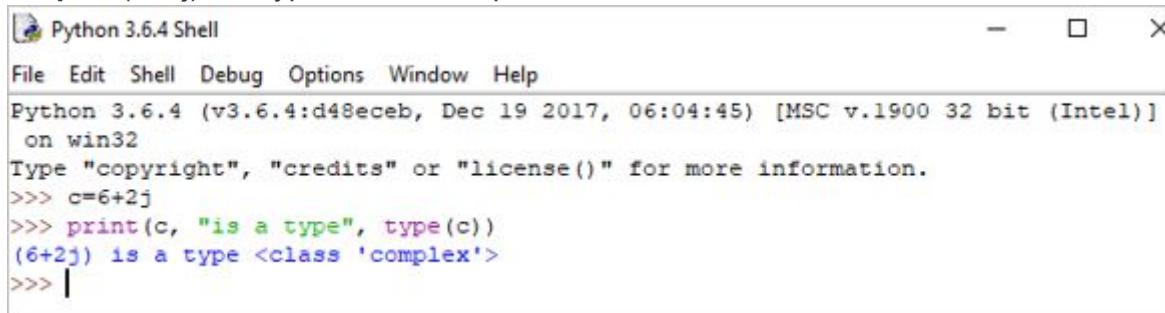
Output: 2.5 is of type <class 'float'>



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> b=2.5
>>> print(b, "is of type", type(b))
2.5 is of type <class 'float'>
>>> |
```

```
1 c = 6+2j
2 print(c, "is a type", type(c))
```

Output: (6+2j) is a type <class 'complex'>



The screenshot shows a Windows-style window titled "Python 3.6.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area displays the Python interpreter's prompt (>>>) followed by the code and its output. The output shows that the variable 'c' is of type 'complex'.

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> c=6+2j
>>> print(c, "is a type", type(c))
(6+2j) is a type <class 'complex'>
>>> |
```

A **complex number** is a **number** that can be expressed in the form $a + bi$, where a and b are real **numbers**, and i is a solution of the equation $x^2 = -1$. Because no real **number** satisfies this equation, i is called an **imaginary number**.

#2) String

A string is an ordered sequence of characters.

We can use single quotes or double quotes to represent strings. Multi-line strings can be represented using triple quotes, "" or """.

Strings are immutable which means once we declare a string we can't update the already declared string.

Example:

```
1 Single = 'Welcome'
2 or
```

```
3 Multi = "Welcome"
```

Multiline: "Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991"

or

```
""Python is an interpreted high-level programming language for general-purpose
programming. Created by Guido van Rossum and first released in 1991.""
```

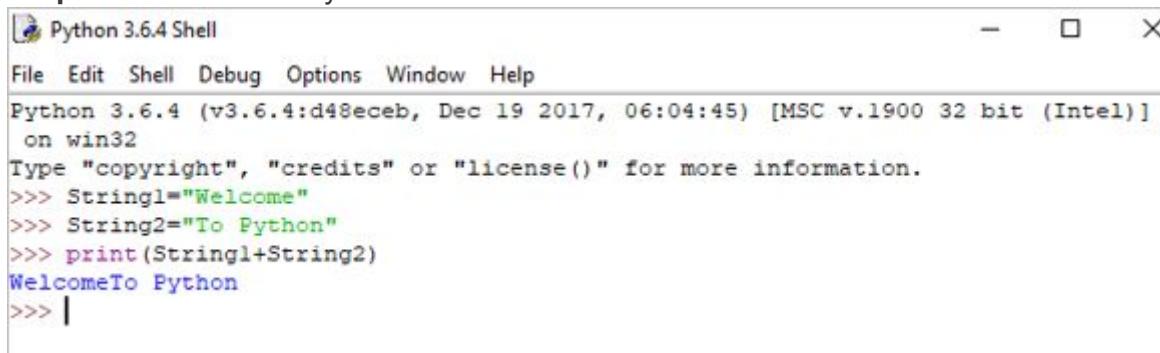
We can perform several operations in strings like **Concatenation**, **Repetition**, and **Slicing**.

Concatenation: It means the operation of joining two strings together.

Example:

```
1 String1 = "Welcome"  
2 String2 ="To Python"  
  
3 print(String1+String2)
```

Output: Welcome To Python



The screenshot shows a Python 3.6.4 Shell window. The title bar says "Python 3.6.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area displays the Python interpreter's prompt (>>>). The user has entered three lines of code: "String1 = 'Welcome'", "String2 = 'To Python'", and "print(String1+String2)". The output is "WelcomeTo Python". The window has standard close, minimize, and maximize buttons in the top right corner.

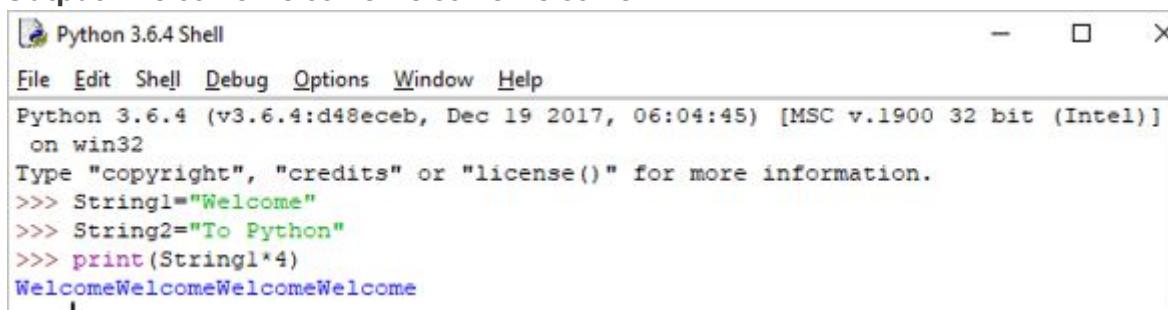
Repetition:

It means repeating a sequence of instructions a certain number of times.

Example:

```
1 Print(String1*4)
```

Output: WelcomeWelcomeWelcomeWelcome



The screenshot shows a Python 3.6.4 Shell window. The title bar says "Python 3.6.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area displays the Python interpreter's prompt (>>>). The user has entered three lines of code: "String1 = 'Welcome'", "String2 = 'To Python'", and "print(String1*4)". The output is "WelcomeWelcomeWelcomeWelcome". The window has standard close, minimize, and maximize buttons in the top right corner.

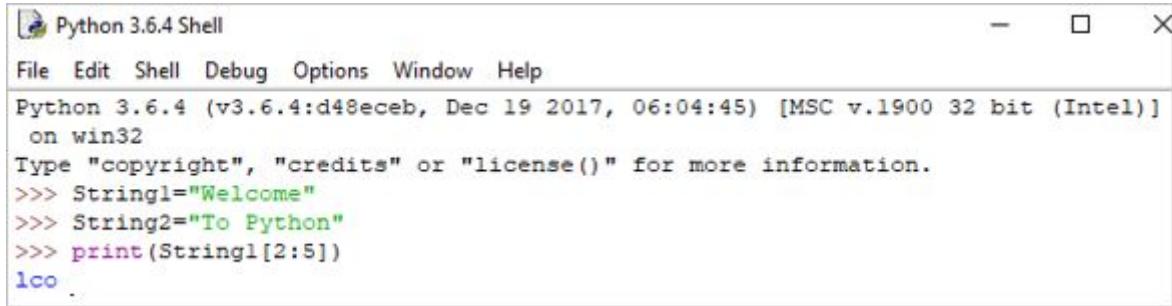
Slicing: Slicing is a technique for extracting parts of a string.

Note: In Python, index starts from 0.

Example:

```
1 print(String1[2:5])
```

Output: lco



Python 3.6.4 Shell

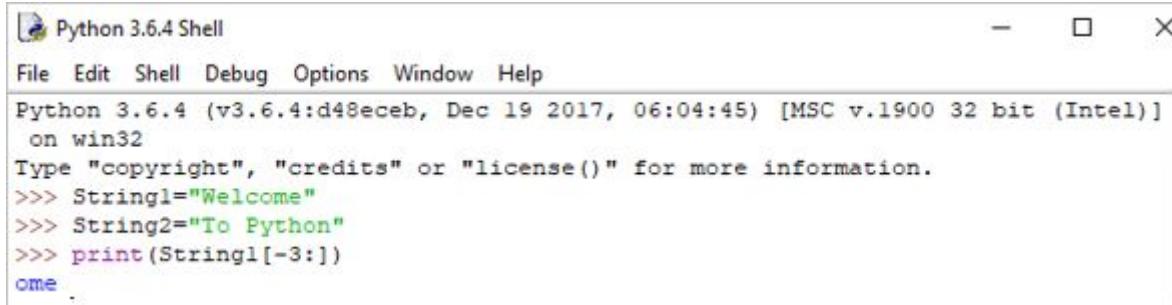
File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> String1="Welcome"
>>> String2="To Python"
>>> print(String1[2:5])
lco
```

Python also supports negative index.

```
1 print(String1[-3:])
```

Output: ome



Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

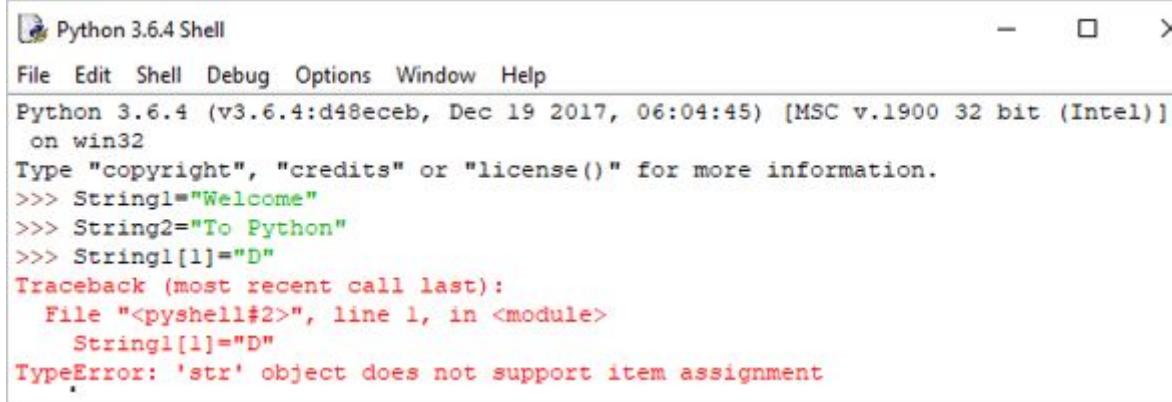
```
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> String1="Welcome"
>>> String2="To Python"
>>> print(String1[-3:])
ome
```

As Strings are immutable in Python, if we try to update the string, then it will generate an error.

Example:

```
1 String[1] = "D"
```

Output: TypeError: 'str' object does not support item assignment



Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> String1="Welcome"
>>> String2="To Python"
>>> String1[1] = "D"
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    String1[1] = "D"
TypeError: 'str' object does not support item assignment
```

#3) List

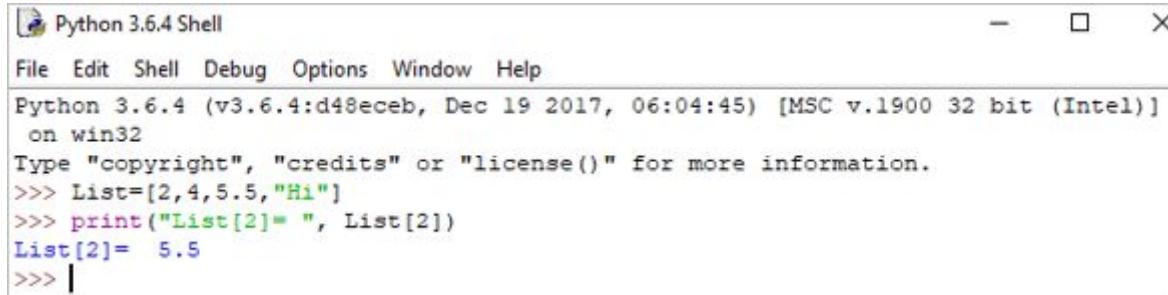
A list can contain a series of values.

List variables are declared by using brackets []. A list is mutable, which means we can modify the list.

Example:

```
1 List = [2,4,5.5,"Hi"]
2 print("List[2] = ", List[2])
```

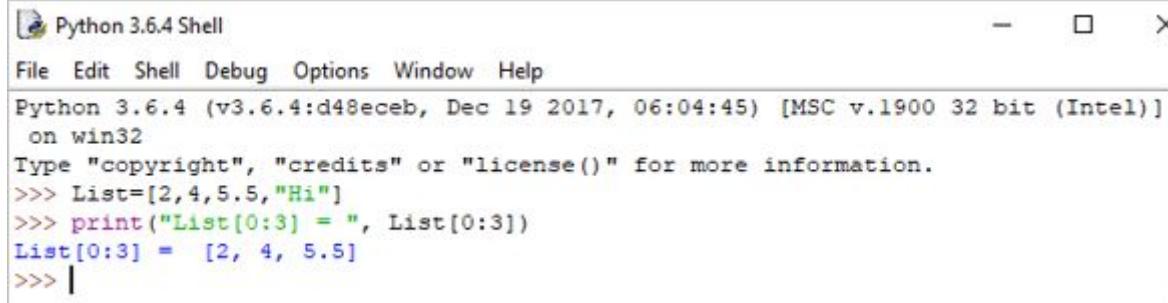
Output: List[2] = 5.5



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> List=[2,4,5.5,"Hi"]
>>> print("List[2]= ", List[2])
List[2]= 5.5
>>> |
```

```
1 print("List[0:3] = ", List[0:3])
```

Output: List[0:3] = [2, 4, 5.5]



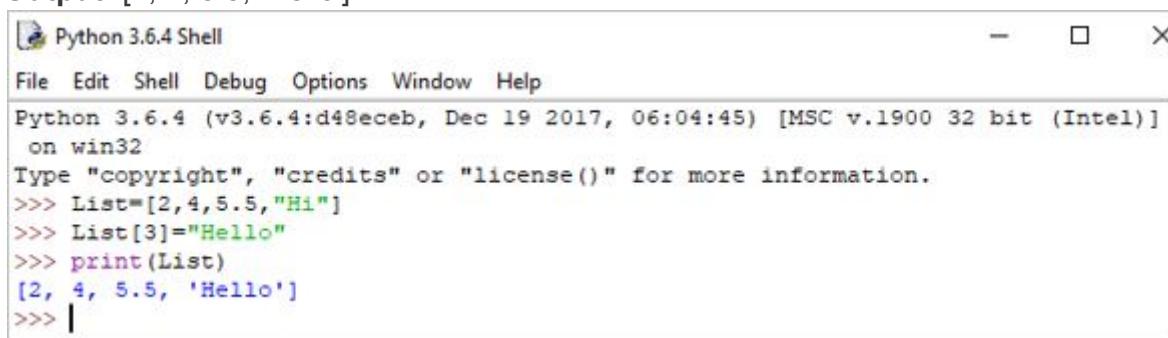
```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> List=[2,4,5.5,"Hi"]
>>> print("List[0:3] = ", List[0:3])
List[0:3] = [2, 4, 5.5]
>>> |
```

Updating the list:

```
1 List[3] = "Hello"
2 If we print the whole list, we can see the updated list.
```

```
3 print(List)
```

Output: [2, 4, 5.5, 'Hello']



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> List=[2,4,5.5,"Hi"]
>>> List[3]="Hello"
>>> print(List)
[2, 4, 5.5, 'Hello']
>>> |
```

#4) Tuple

A tuple is a sequence of Python objects separated by commas.

Tuples are immutable, which means tuples once created cannot be modified. Tuples are defined using parentheses () .

Example:

```
1 Tuple = (50,15,25.6,"Python")
2 print("Tuple[1] = ", Tuple[1])
```

Output: Tuple[1] = 15

A screenshot of the Python 3.6.4 Shell window. The title bar says "Python 3.6.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area shows the Python interpreter's prompt (>>>). The code entered is: "Tuple = (50,15,25.6,"Python")", "print("Tuple[1] = ", Tuple[1])", and "Tuple[1] = 15". The output is: "Tuple[1] = 15".

```
1 print("Tuple[0:3] =", Tuple[0:3])
```

Output: Tuple[0:3] = (50, 15, 25.6)

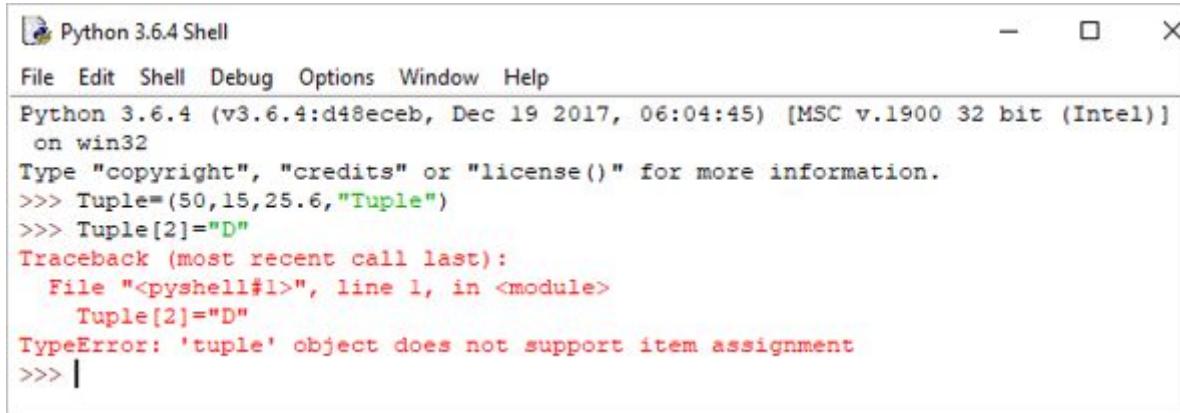
A screenshot of the Python 3.6.4 Shell window. The title bar says "Python 3.6.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area shows the Python interpreter's prompt (>>>). The code entered is: "Tuple = (50,15,25.6,"Python")", "print("Tuple[0:3] = ", Tuple[0:3])", and "Tuple[0:3] = (50, 15, 25.6)". The output is: "Tuple[0:3] = (50, 15, 25.6)".

As Tuples are immutable in Python, if we try to update the tuple, then it will generate an error.

Example:

```
1 Tuple[2]= "D"
```

Output: TypeError: 'tuple' object does not support item assignment



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Tuple=(50,15,25.6,"Tuple")
>>> Tuple[2]="D"
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    Tuple[2]="D"
TypeError: 'tuple' object does not support item assignment
>>> |
```

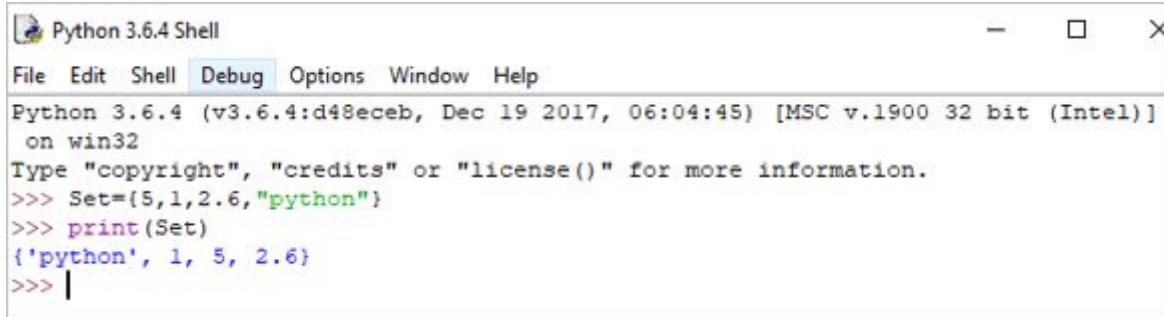
#5) Set

A set is an unordered collection of items. Set is defined by values separated by a comma inside braces {}.

Example:

```
1 Set = {5,1,2.6,"python"}
2 print(Set)
```

Output: {'python', 1, 5, 2.6}



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Set={5,1,2.6,"python"}
>>> print(Set)
{'python', 1, 5, 2.6}
>>> |
```

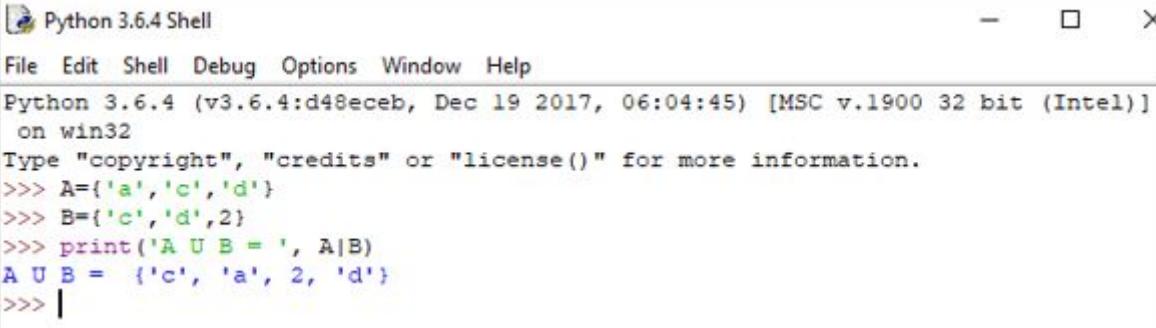
In the set, we can perform operations like **union** and **intersection** on two sets.

We can perform Union operation by Using | Operator.

Example:

```
1 A = {'a', 'c', 'd'}
2 B = {'c', 'd', 2 }
3 print('A U B =', A| B)
```

Output: A U B = {'c', 'a', 2, 'd'}



Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

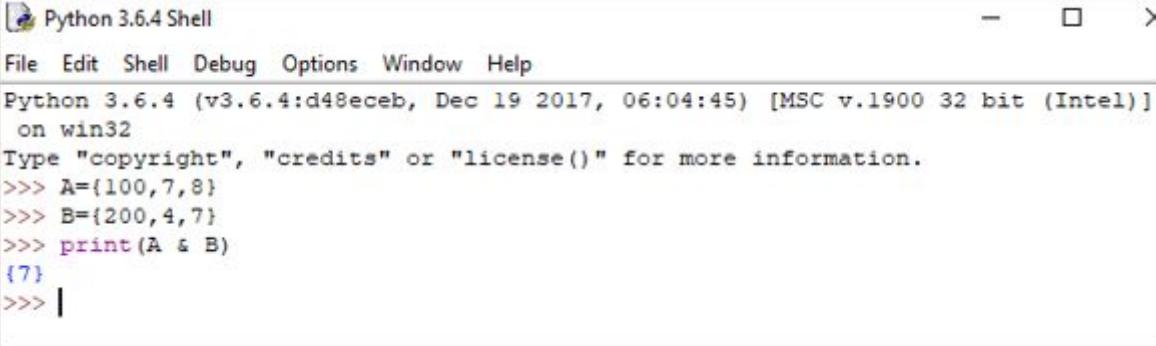
```
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> A={'a','c','d'}
>>> B={'c','d',2}
>>> print('A U B = ', A|B)
A U B =  {'c', 'a', 2, 'd'}
>>> |
```

We can perform Intersection operation by Using & Operator.

```
1 A = {100, 7, 8}
2 B = {200, 4, 7}

3 print(A & B)
```

Output: {7}



Python 3.6.4 Shell

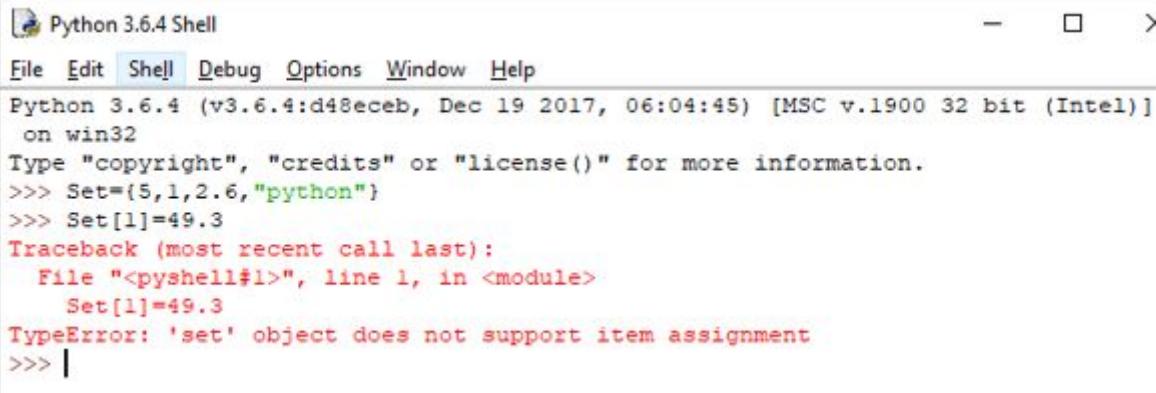
File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> A={100,7,8}
>>> B={200,4,7}
>>> print(A & B)
{7}
>>> |
```

As the set is an unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

```
1 Set[1] = 49.3
```

Output: TypeError: 'set' object does not support item assignment



Python 3.6.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Set={5,1,2.6,"python"}
>>> Set[1]=49.3
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    Set[1]=49.3
TypeError: 'set' object does not support item assignment
>>> |
```

#6) Dictionary

Dictionaries are the most flexible built-in data type in python.

Dictionaries items are stored and fetched by using the key. Dictionaries are used to store a huge amount of data. To retrieve the value we must know the key. In Python, dictionaries are defined within braces {}.

We use the key to retrieve the respective value. But not the other way around.

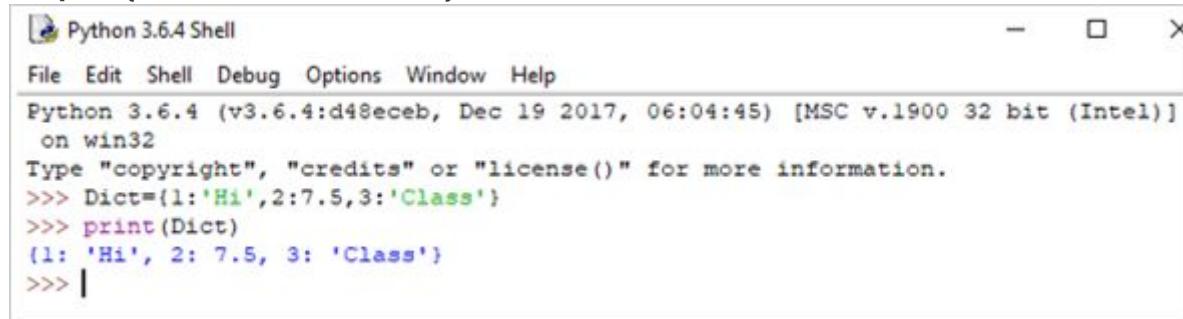
Syntax:

Key:value

Example:

```
1 Dict = {1:'Hi',2:7.5, 3:'Class'}
2 print(Dict)
```

Output: {1: 'Hi', 2: 7.5, 3: 'Class'}



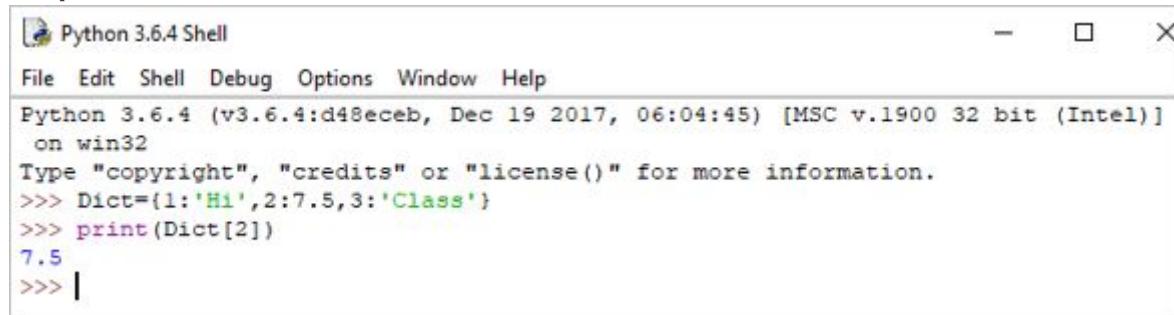
```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> print(Dict)
{1: 'Hi', 2: 7.5, 3: 'Class'}
>>> |
```

We can retrieve the value by using the following method:

Example:

```
1 print(Dict[2])
```

Output: 7.5



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48ebeb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> print(Dict[2])
7.5
>>> |
```

If we try to retrieve the value by using the value instead of the key, then it will generate an error.

Example:

```
1 print("Dict[7.5] = ", Dict[7.5])
```

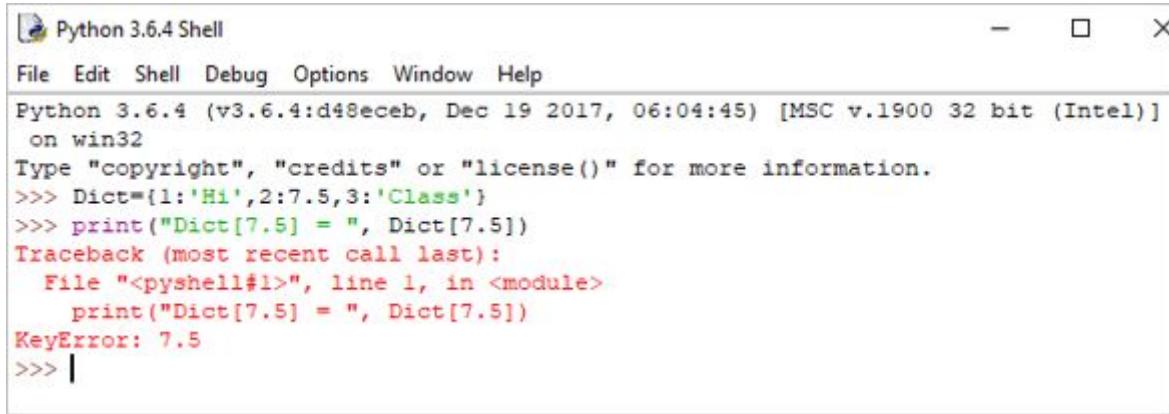
Output:

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

```
print("Dict[7.5] = ", Dict[7.5])
```

KeyError: 7.5



The screenshot shows a Python 3.6.4 Shell window. The command line shows the creation of a dictionary with keys 1, 2, and 3, and values 'Hi', 7.5, and 'Class' respectively. When the print statement is run, it tries to access the key 7.5, which does not exist, resulting in a KeyError: 7.5.

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> print("Dict[7.5] = ", Dict[7.5])
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print("Dict[7.5] = ", Dict[7.5])
KeyError: 7.5
>>> |
```

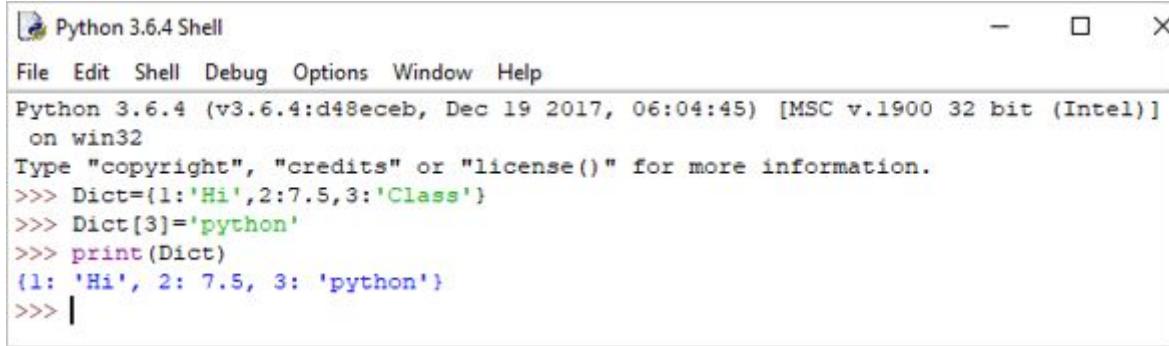
We can update the dictionary by using the following methods as well:

Example:

```
1 Dict[3] = 'python'
2 print(Dict)
```

Output:

```
{1: 'Hi', 2: 7.5, 3: 'python'}
```



The screenshot shows a Python 3.6.4 Shell window. The command line shows the creation of a dictionary with keys 1, 2, and 3, and values 'Hi', 7.5, and 'python' respectively. The print statement then outputs the entire dictionary, showing that the value for key 3 has been successfully updated.

```
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> Dict={1:'Hi',2:7.5,3:'Class'}
>>> Dict[3]='python'
>>> print(Dict)
{1: 'Hi', 2: 7.5, 3: 'python'}
>>> |
```

Hope you must have understood the various classifications of Python Data Types by now, from this tutorial.