Advanced Programming
Topic 5: Very Brief Introduction to Python

Nguyễn, Thiện Bình
Mechatronics and Sensor Systems Technology
Vietnamese – German University

19 January 2025

Contents

Basics of Python

Installation:

1. **Anaconda** platform
2. Install a new package: *conda install jupyter*
3. Create a new environment: *conda create –name myenv python=3.10.9*
4. Activate an environment: *conda activate myenv*
5. Run Jupyter: *jupyter notebook*

Basics of Python

Python:

1. Interpreted language (codes run line/cell by line/cell)
2. Implicit coding (no rigorous declarations needed)
3. Powerful on mixed-type data structures
4. Strongly OOP

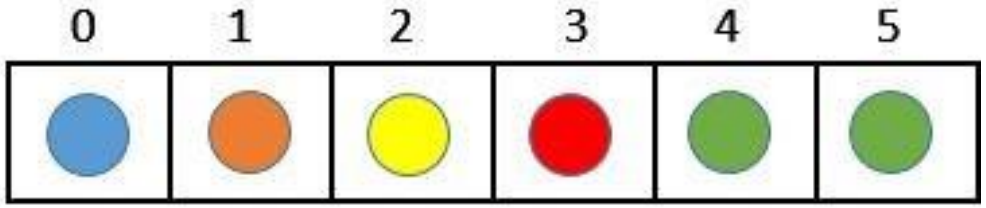
Data Structures

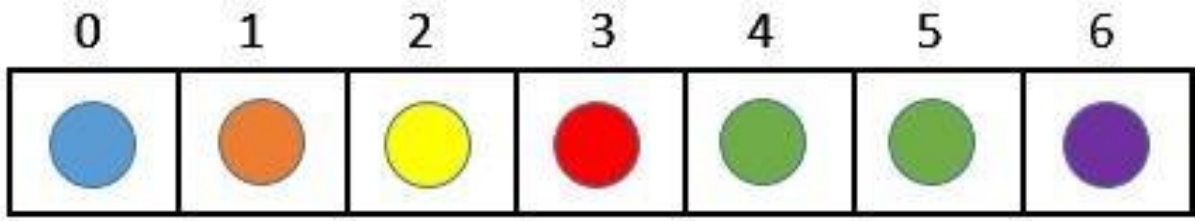
1. Lists
2. Tuples
3. Sets
4. Dictionaries

Data Structures: LISTS

What is a list?

- Mutable / modifiable
- Ordered (indexed elements)

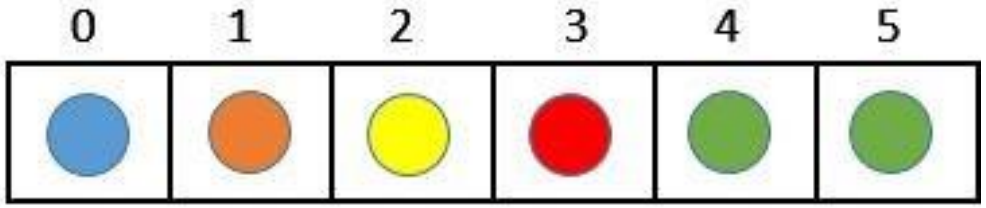




Data Structures: LISTS

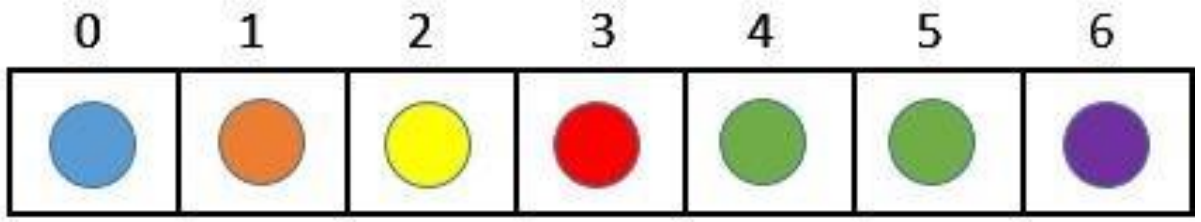
What is a list?

- Mutable / modifiable
- Ordered (indexed elements)



In Python:

- A list is created using square brackets []



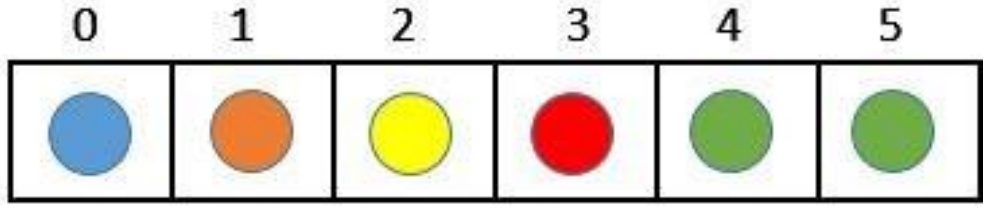

```
### List ###
list1 = [1, 2, 3, 4, 5]
print('list1 = ', list1)
```

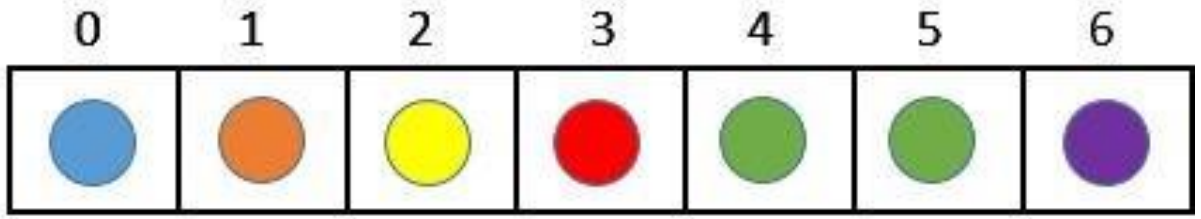
```
list1 =  [1, 2, 3, 4, 5]
```

Data Structures: LISTS

In Python:

- **A list is ordered:** element duplication is allowed






```
### list ###
### duplicate elements are OK ###
list1 = [1, 1, 2, 'Henry', 'Henry']
print('list1 = ', list1)
```

```
list1 =  [1, 1, 2, 'Henry', 'Henry']
```

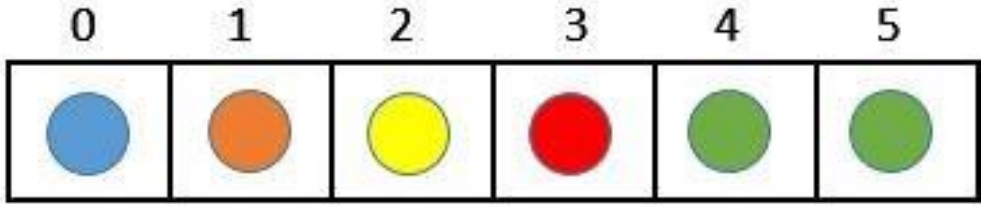
Data Structures: LISTS

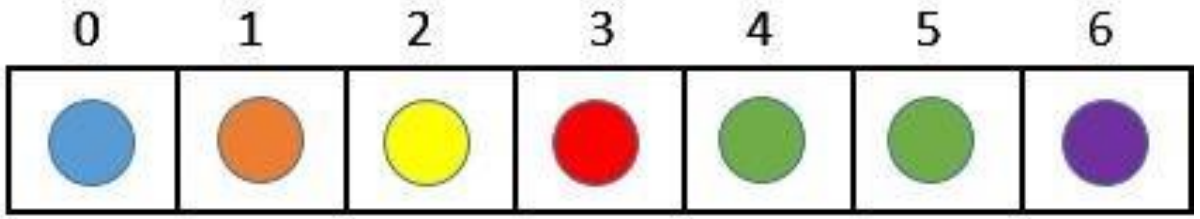
In Python:

- **A list is ordered:** element access via index

```
### List ###
list1 = [1, 2, 3, 4, 5]
print('list1 = ', list1)
# Ordered element accessed by index
for i in range(5):
    print('list1[%d] = %d' % (i, list1[i]))
```

```
list1 =  [1, 2, 3, 4, 5]
list1[0] = 1
list1[1] = 2
list1[2] = 3
list1[3] = 4
list1[4] = 5
```





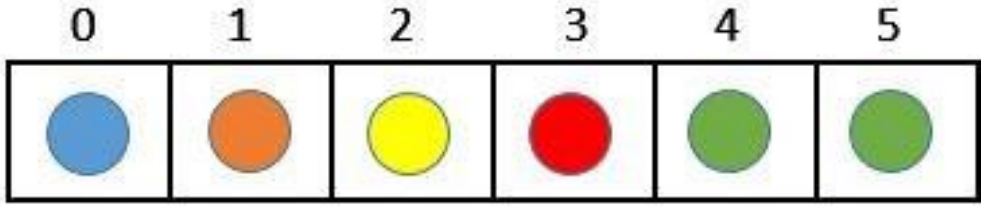
Data Structures: LISTS

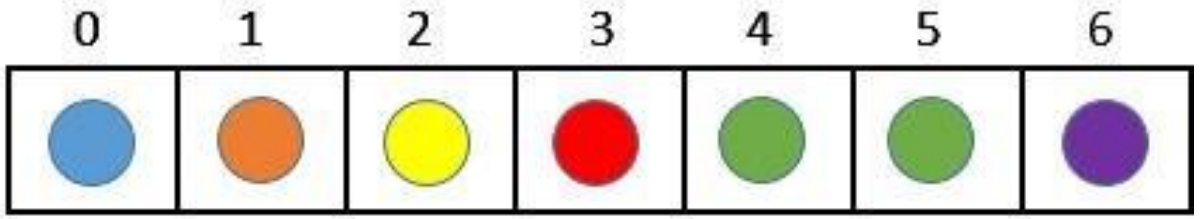
In Python:

- **A list is ordered:** for each loop

```
### List ###
list1 = [1, 2, 3, 4, 5]
# for each looping
for element in list1:
    print(element, end = ", ")
```

```
1, 2, 3, 4, 5,
```





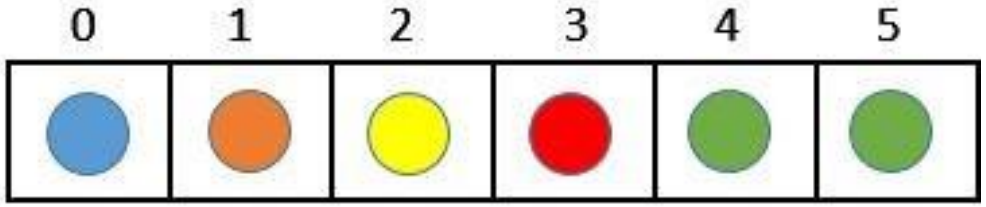
Data Structures: LISTS

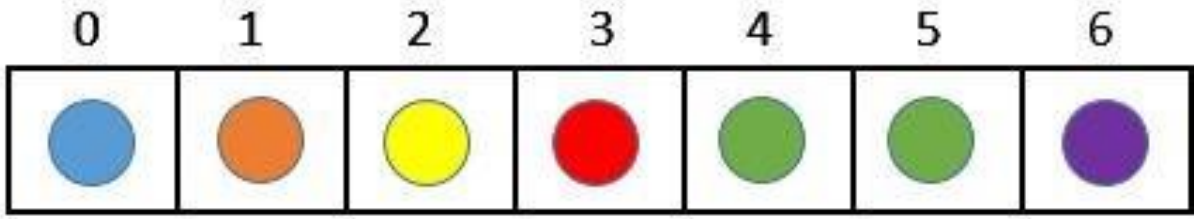
In Python:

- **A list is mutable:** can be modified after created

```
### list ###
list1 = [1, 2, 3, 4, 5]
print('list1 = ', list1)
# mutable: changing elements
list1[1] = 'Henry'
print('list1 = ', list1)
# mutable: adding new element
list1.append('True')
print('list1 = ', list1)
```

```
list1 =  [1, 2, 3, 4, 5]
list1 =  [1, 'Henry', 3, 4, 5]
list1 =  [1, 'Henry', 3, 4, 5, 'True']
```

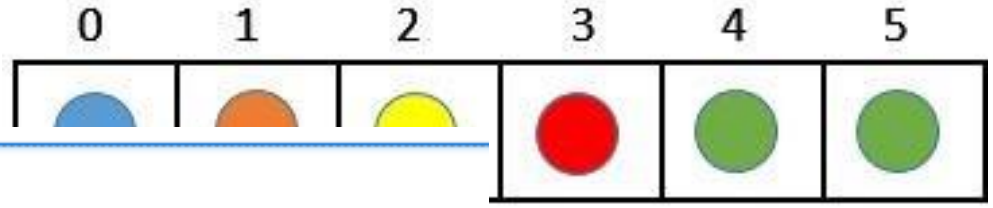




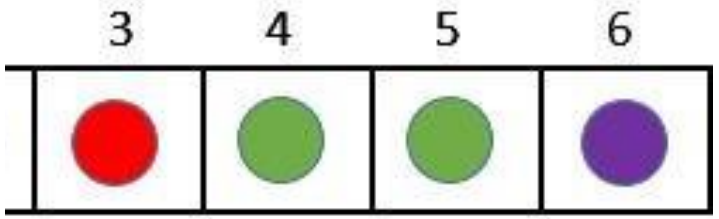
Data Structures: LISTS

In Python:

- **List methods:**




```
### list ###
list1 = [3, 'Henry', 3, 'True', 5]
len(list1)                  # get the list length
list1.index('Henry')        # get the element index
list1.count(3)              # count occurrence of value 3
list1.append(-2)            # append new element to the end
list1.pop(0)                # remove element at index 0
list1.insert(2,[10, -10]) # insert element at index 2
```

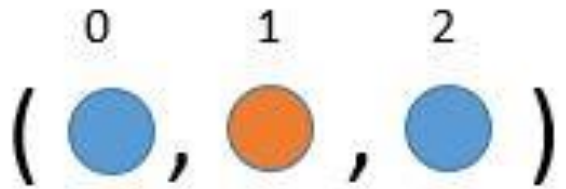



```
help(list)
```

Data Structures: TUPLES

What is a tuple?

- Immutable / unmodifiable
- Ordered (indexed elements)

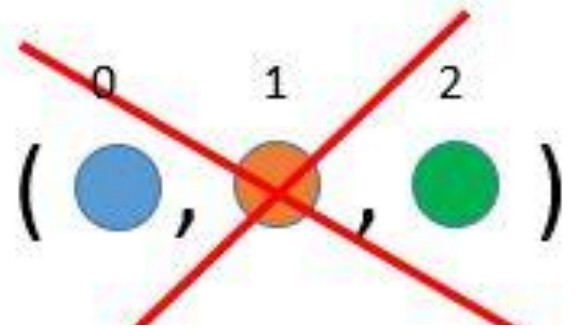


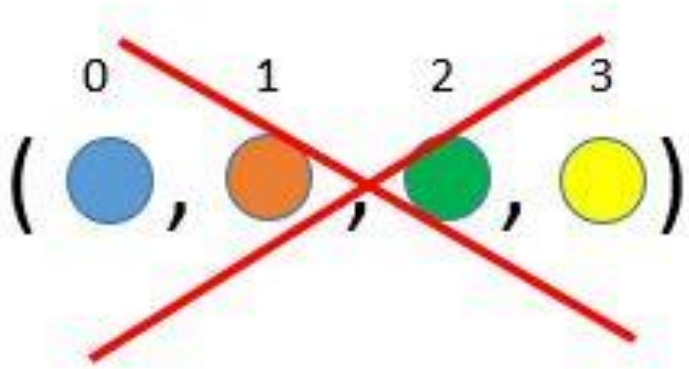
In Python:

- A tuple is created using parentheses ()

```
### a tuple ###
tuple1 = (1, 2, 'Henry')
print(tuple1)
```

```
(1, 2, 'Henry')
```

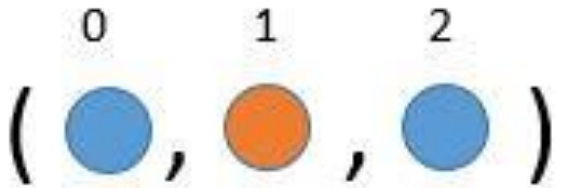




Data Structures: TUPLES

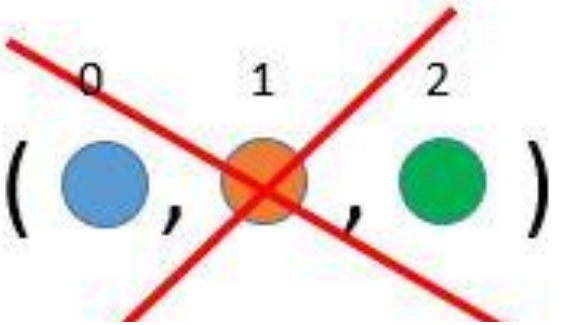
In Python:

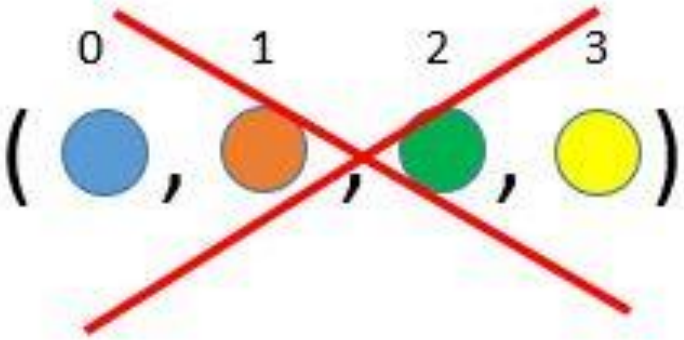
- **A tuple is ordered:** element access via index




```
### a tuple ###
tuple1 = (1, 2, 'Henry')
# access via index
for i in range(3):
    print('element[%d] = ' % i + '{0}'.format(tuple1[i]))
```

```
element[0] = 1
element[1] = 2
element[2] = Henry
```



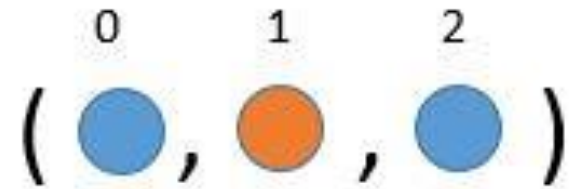


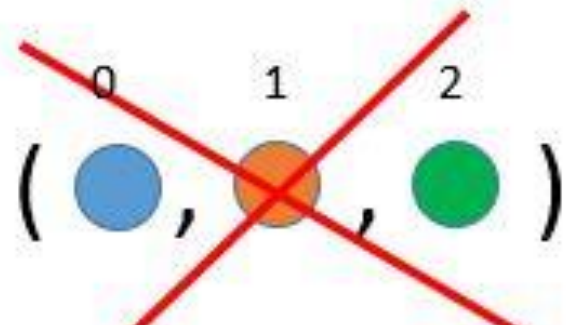
Data Structures: TUPLES

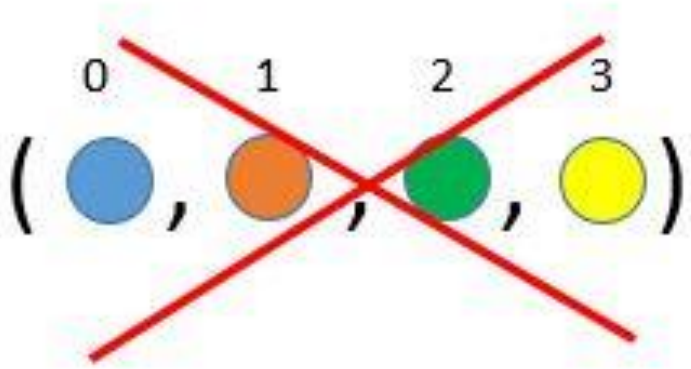
In Python:

- **A tuple is immutable:**
  modifications not allowed

```
### a tuple ###
tuple1 = (1, 2, 'Henry')
# modifications not allowed
#tuple1[0] = 100
#tuple1.append('True')
```



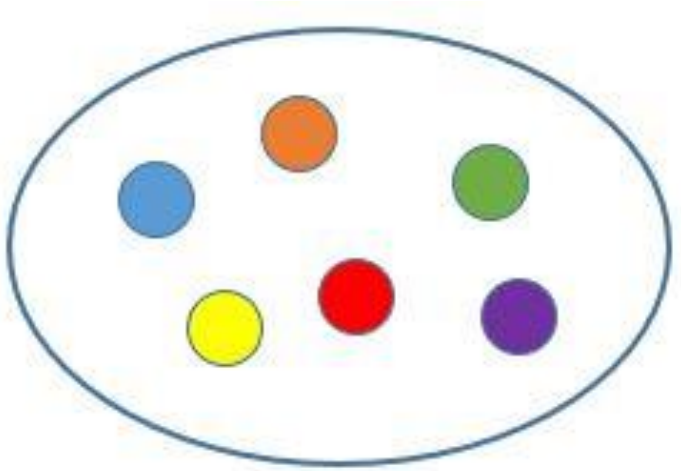




Data Structures: SETS

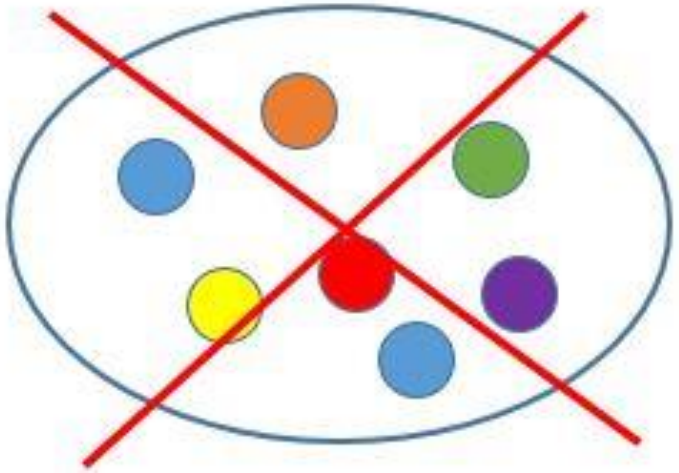
What is a set?

- Unique elements (duplication not allowed)
- Unordered (no index system)



In Python:

- A set is created using curly brackets {}



```
### a set ###
set1 = {1, 2, 'Henry', 'True'}
print(set1)
```

```
{1, 2, 'True', 'Henry'}
```

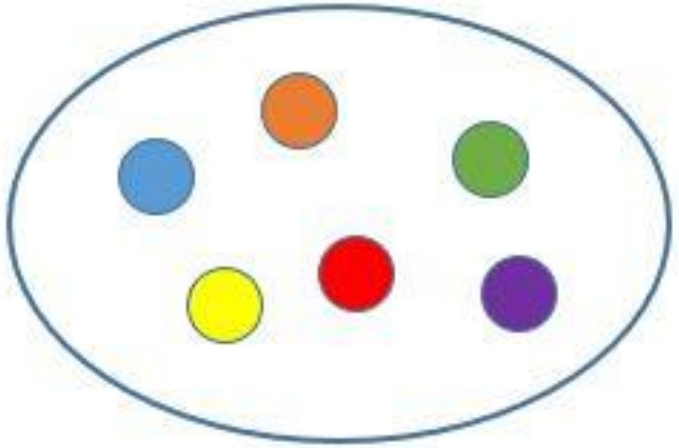
Data Structures: SETS

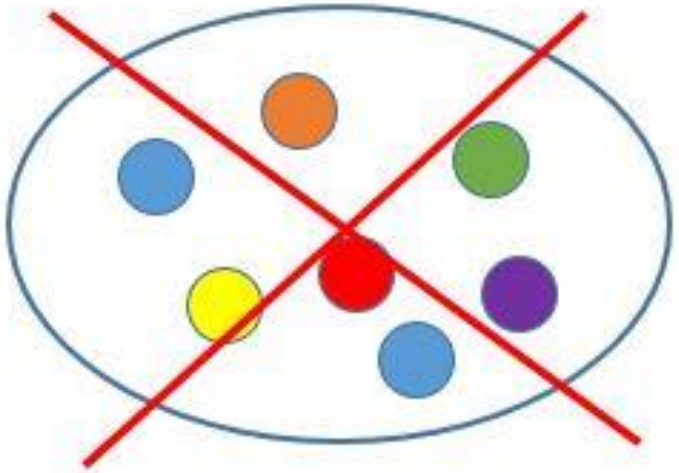
In Python:

- **A set is unordered:** element access via for each loop

```
### a set ###
set1 = {1, 2, 'Henry', 'True'}
# element access via for each loop
for element in set1:
    print('element = {}'.format(element))
```

```
element = 1
element = 2
element = True
element = Henry
```





Data Structures: SETS

In Python:

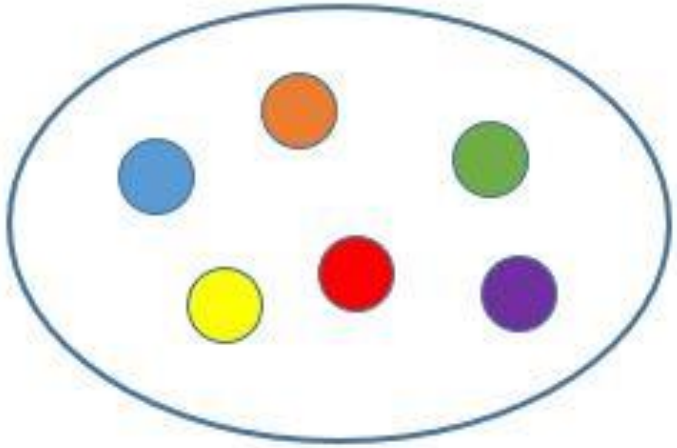
- **No duplication:** set elements are unique

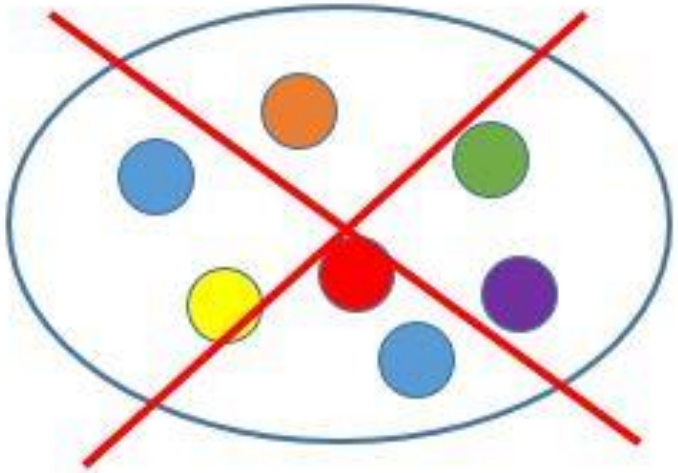
```
### a set ###
set1 = {1, 2, 'Henry', 'True'}
print(set1)
set1.add('Henry')  # duplication not allowed
print(set1)
```

```
{1, 2, 'True', 'Henry'}
{1, 2, 'True', 'Henry'}
```

```
### a set ###
set1 = {1, 1, 1, 2, 'Henry', 'True'}
print(set1)   # duplicate elements are removed
```

```
{1, 2, 'True', 'Henry'}
```





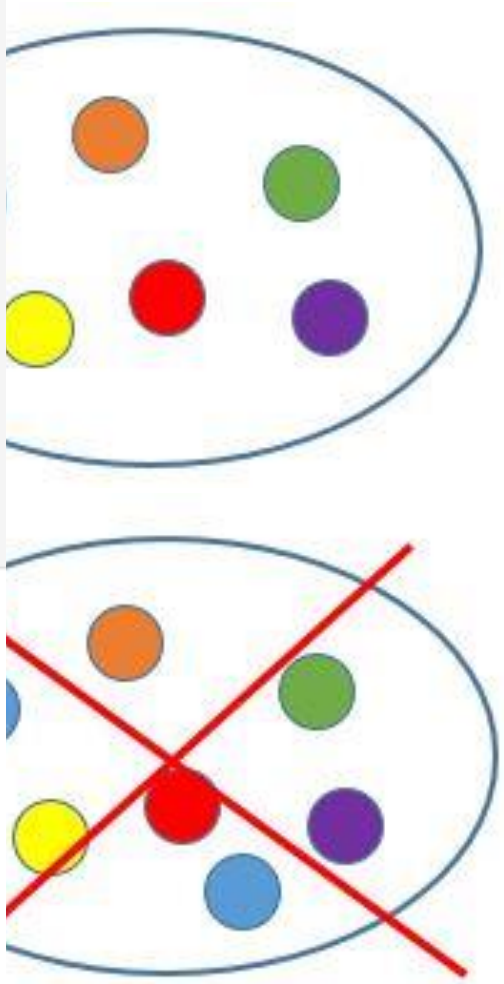
Data Structures: SETS

In Python:

- **Set methods:**

```
### set methods ###
set1 = {1, 2, 'Henry', 'Jessica'}
print('set1 = ', set1)
set2 = {10, 2, 'Jessica', 'John'}
print('set2 = ', set2)
print('Union(set1, set2) = ', set1 | set2)
print('Intersection(set1, set2) = ', set1 & set2)
print('Difference(set1, set2) = ', set1 - set2)
print('Difference(set2, set1) = ', set2 - set1)
print('Symmetric Difference(set1, set2) = ', set1 ^ set2)
print('Symmetric Difference(set2, set1) = ', set2 ^ set1)
```

```
set1 =  {1, 2, 'Jessica', 'Henry'}
set2 =  {10, 2, 'Jessica', 'John'}
Union(set1, set2) =  {1, 2, 10, 'Henry', 'Jessica', 'John'}
Intersection(set1, set2) =  {2, 'Jessica'}
Difference(set1, set2) =  {1, 'Henry'}
Difference(set2, set1) =  {10, 'John'}
Symmetric Difference(set1, set2) =  {1, 10, 'Henry', 'John'}
Symmetric Difference(set2, set1) =  {1, 10, 'Henry', 'John'}
```



```
help(set)
```

Data Structures: DICTIONARIES

What is a dictionary?

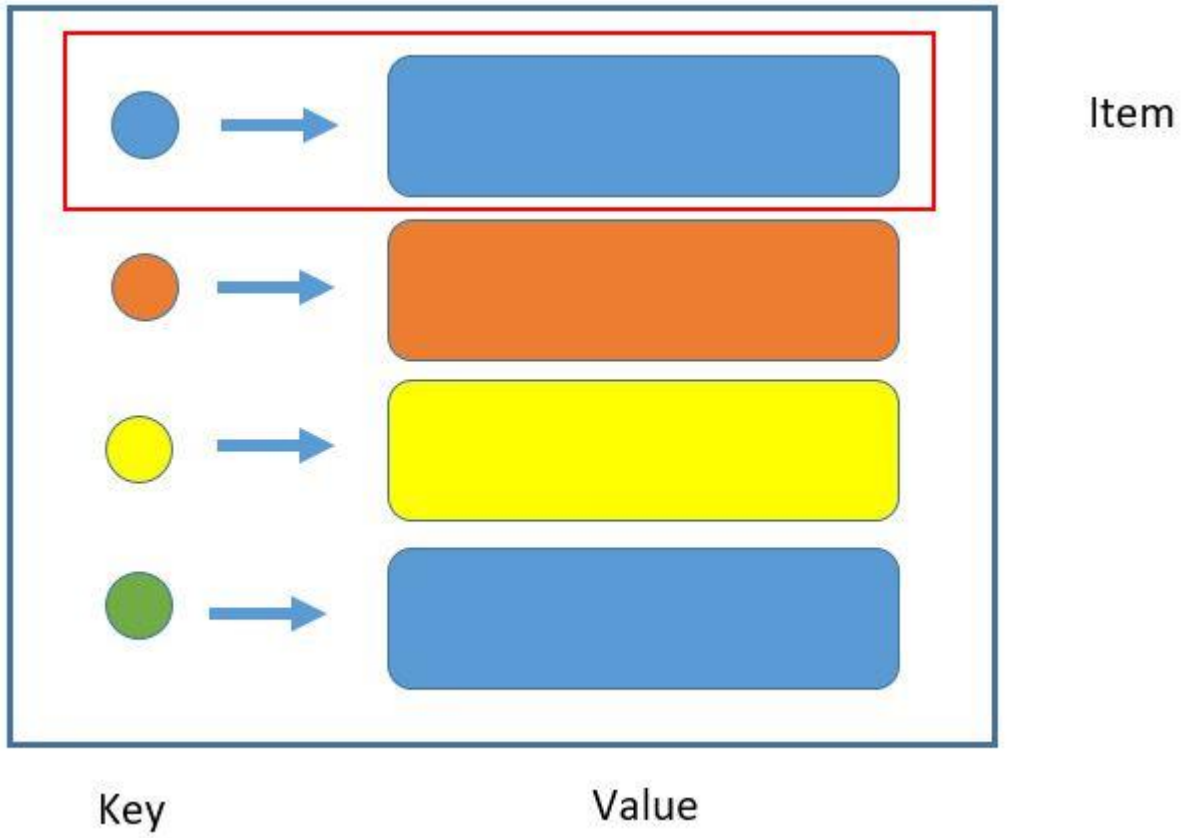
- Unordered items of key – value pairs
- Keys are **unique**

In Python:

- A dictionary is created using curly brackets **{}** in which the key and value of a pair is separated by a colon **:**

```
### a dictionary ###
dict1 = {1:'Henry', 2: 'Jessica', 3: 'John'}
print(dict1)
```

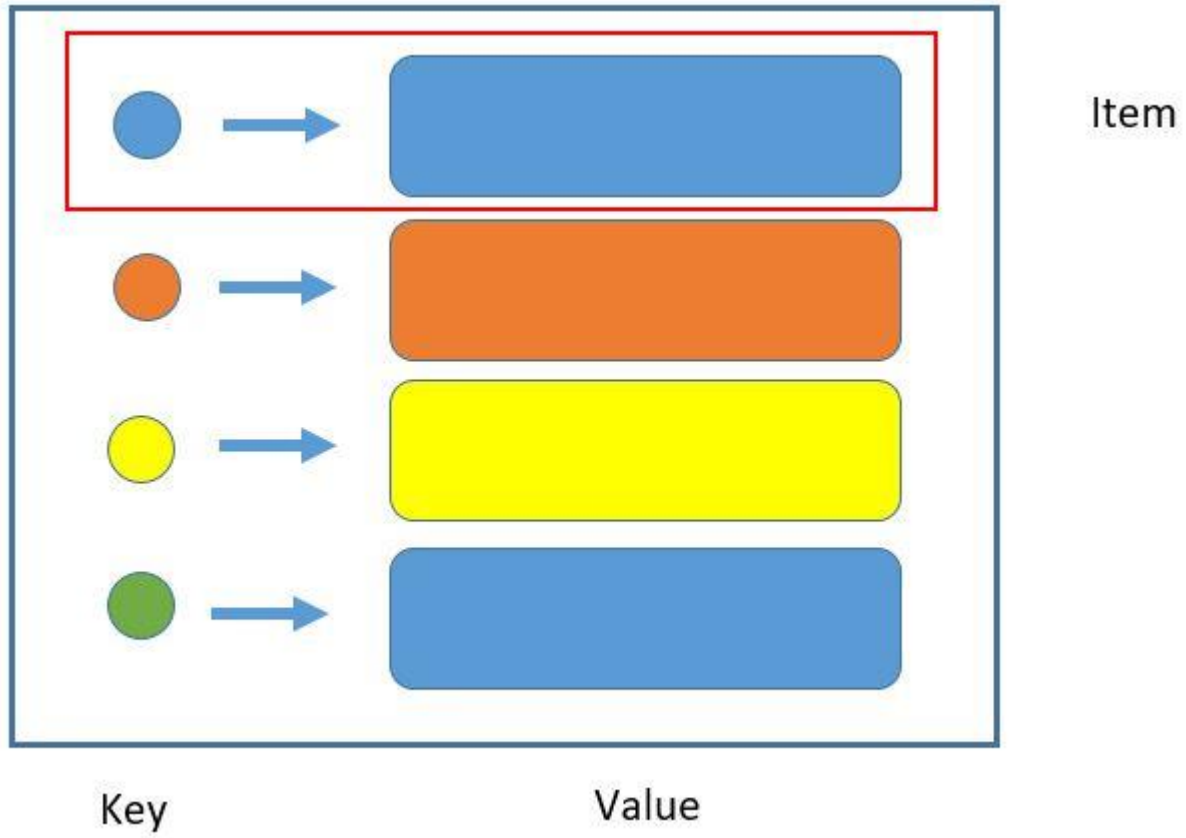
```
{1: 'Henry', 2: 'Jessica', 3: 'John'}
```



Data Structures: DICTIONARIES

In Python:

- Item access via unique keys



Data Structures: DICTIONARIES

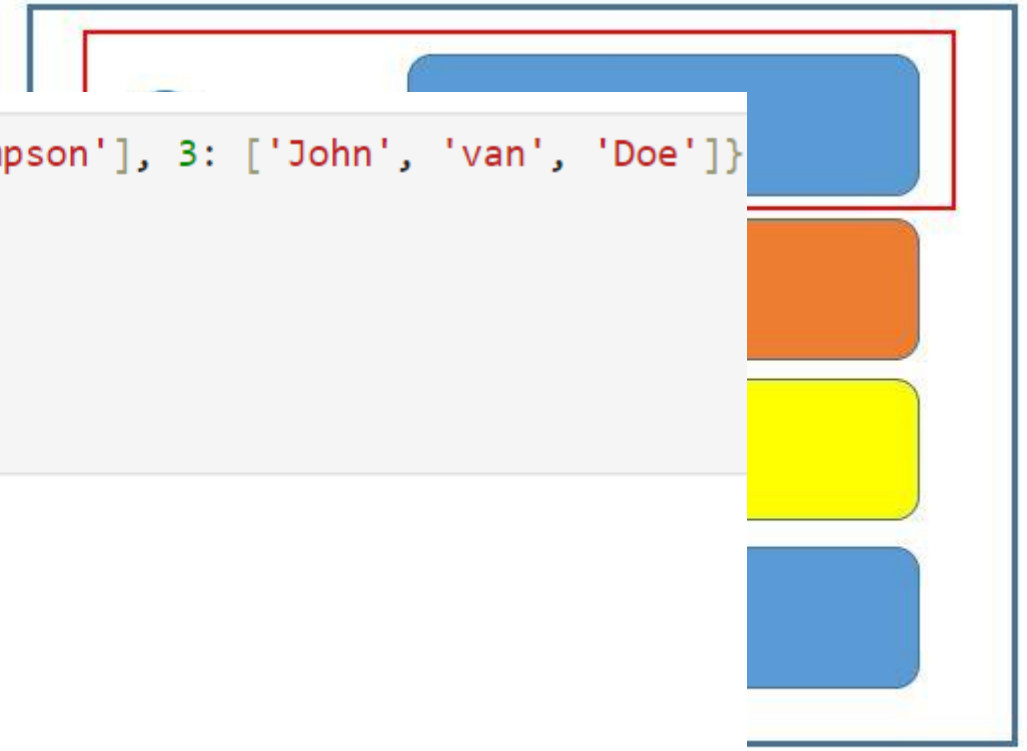
In Python:

- Item access via unique keys

```
dict1 = {1:['Henry','Graham'], 2: ['Jessica', 'Thompson'], 3: ['John', 'van', 'Doe']}
for key in dict1:
    value = dict1.get(key)
    print('Item: ')
    print('  --> key = ', key)
    print('  --> value = ', *value)
```

```
Item:
  --> key =  1
  --> value =  Henry Graham
Item:
  --> key =  2
  --> value =  Jessica Thompson
Item:
  --> key =  3
  --> value =  John van Doe
```

```
help(dict)
```



Functions, Classes and Objects

1. Functions
2. Classes and Objects
3. Inheritance

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
24
```

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
```

```
24
```

```
### a function ###
def sayHi(name):
    """ Print a greeting to an inputted name """
    print(f'Greetings, {name}!')

sayHi('Henry')
dummy = sayHi('George')
```

```
Greetings, Henry!
Greetings, George!
```

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
24
```

return product

```
### a function ###
def sayHi(name):
    """ Print a greeting to an inputted name """
    print(f'Greetings, {name}!')

sayHi('Henry')
dummy = sayHi('George')
```

```
Greetings, Henry!
Greetings, George!
```

return NonType

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
24
```

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
24
```

```
help(multiply)
```

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
24
```

```
help(multiply)
```

```
Help on function multiply in module __main__:

multiply(a, b)
    return the product of a times b
```

Functions

- A function is defined by the '**def**' keyword

```
### a function ###
def function_name(input arguments):
    # function body
    return result
```

```
### a function ###
def multiply(a, b):
    """ return the product of a times b """
    product = a*b
    return product

a = multiply(5, 10)
print(a)
multiply(3, 8)
```

```
50
24
```

```
help(multiply)
```

```
Help on function multiply in module __main__:

multiply(a, b)
    return the product of a times b
```

Use the triple quotation for function documentation

Functions

Pointer-like operator *

- Points to a data structure of an arbitrary number of elements, used as a function input

```
### pointer-like * operator for 1D array ###
def sum2(a, b):
    return a + b

def sum3(a, b, c):
    return a + b + c

def sumArbi(*array):
    result = 0.0
    for element in array:
        result = result + element
    return result
```

```
print(sumArbi(1, 2, 3, 4, 5))
```

```
15.0
```

Functions

Pointer-like operator *

- Points to a data structure of an arbitrary number of elements, used as a function input

```
def dictCheck(**dict):
  for item in dict:
    print(f"{dict[item]} is a {item}")

dict = {'gas' : 'CO2', 'liquid' : 'water', 'solid' : 'iron'}
dictCheck(**dict)
```

```
CO2 is a gas
water is a liquid
iron is a solid
```

Classes and Instances (Objects)

1. Definition
2. Encapsulation
3. Method Overloading
4. Inheritance and Polymorphism

Classes and Instances (Objects)

Class definition:

- Keyword: **class**
- Constructor: **__init__**
- *“this”* object: **self**
- Attributes: **self.attribute**
- Methods: **def**

Classes and Instances (Objects)

Class definition:

- Keyword: **class**
- Constructor: **__init__**
- *"this"* object: **self**
- Attributes: **self.attribute**
- Methods: **def**

```
class Point2D:
    def __init__(self, x_coord, y_coord):
        self.x_coord = x_coord
        self.y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.x_coord}')
        print(f'  --> y-coord = {self.y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
```

Classes and Instances (Objects)

Class definition:

- Keyword: **class**
- Constructor: **__init__**
- *"this"* object: **self**
- Attributes: **self.attribute**
- Methods: **def**

```python
class Point2D:
    def __init__(self, x_coord, y_coord):
        self.x_coord = x_coord
        self.y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.x_coord}')
        print(f'  --> y-coord = {self.y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
```

```
2D point:
  --> x-coord = 3.0
  --> y-coord = 5.0
```

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

```python
class Point2D:
    # public attributes
    x_coord = 0.0
    y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.x_coord = x_coord
        self.y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.x_coord}')
        print(f'  --> y-coord = {self.y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
```

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

```
2D point:
  --> x-coord = 3.0
  --> y-coord = 5.0
```

```python
class Point2D:
    # public attributes
    x_coord = 0.0
    y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.x_coord = x_coord
        self.y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.x_coord}')
        print(f'  --> y-coord = {self.y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
```

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

```
class Point2D:
    # public attributes
    x_coord = 0.0
    y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.x_coord = x_coord
        self.y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.x_coord}')
        print(f'  --> y-coord = {self.y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
## changing the coordinates
pt.x_coord = 30.0
pt.y_coord = 50.0
pt.print()
```

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

```
2D point:
  --> x-coord = 3.0
  --> y-coord = 5.0
2D point:
  --> x-coord = 30.0
  --> y-coord = 50.0
```

```python
class Point2D:
    # public attributes
    x_coord = 0.0
    y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.x_coord = x_coord
        self.y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.x_coord}')
        print(f'  --> y-coord = {self.y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
## changing the coordinates
pt.x_coord = 30.0
pt.y_coord = 50.0
pt.print()
```

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

```
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
## changing the coordinates
pt.__x_coord = 30.0
pt.__y_coord = 50.0
pt.print()
```

Classes and Instances (Objects)

Encapsulation:

- **Public:** no hyphens before names
- **Protected:** ONE hyphen before names
- **Private:** TWO hyphens before names

```
2D point:
  --> x-coord = 3.0
  --> y-coord = 5.0
2D point:
  --> x-coord = 3.0
  --> y-coord = 5.0
```

```python
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
## changing the coordinates
pt.__x_coord = 30.0
pt.__y_coord = 50.0
pt.print()
```

Classes and Instances (Objects)

Method Overloading:

- Optional input: **= None**

```python
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def print(self, x_coord=None, y_coord=None):
        if x_coord is not None and y_coord is not None:
            self.__x_coord = x_coord
            self.__y_coord = y_coord
        else:
            if x_coord is not None:
                self.__x_coord = x_coord
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
pt.print(30.0)          # overloaded print method
pt.print(300.0, 500.0) # overloaded print method
```

Classes and Instances (Objects)

Method Overloading:

- Optional input: **= None**

```
2D point:
  --> x-coord = 3.0
  --> y-coord = 5.0
2D point:
  --> x-coord = 30.0
  --> y-coord = 5.0
2D point:
  --> x-coord = 300.0
  --> y-coord = 500.0
```

```python
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def print(self, x_coord=None, y_coord=None):
        if x_coord is not None and y_coord is not None:
            self.__x_coord = x_coord
            self.__y_coord = y_coord
        else:
            if x_coord is not None:
                self.__x_coord = x_coord
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt = Point2D(3.0, 5.0)
pt.print()
pt.print(30.0)          # overloaded print method
pt.print(300.0, 500.0) # overloaded print method
```

Classes and Instances (Objects)

Operator Overloading:

- Overload the print method: **__str__(self)**

```
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def __str__(self):   # overload the print method
        return f'2D point:\n \
        --> x-coord = {self.__x_coord}\n \
        --> y-coord = {self.__y_coord}'

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt = Point2D(3.0, 5.0)
print(pt)     # overloaded Python's print method
```

Classes and Instances (Objects)

Operator Overloading:

- Overload the print method: **__str__(self)**

```
2D point:
        --> x-coord = 3.0
        --> y-coord = 5.0
```

```python
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def __str__(self):   # overload the print method
        return f'2D point:\n \
        --> x-coord = {self.__x_coord}\n \
        --> y-coord = {self.__y_coord}'

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt = Point2D(3.0, 5.0)
print(pt)     # overloaded Python's print method
```

Classes and Instances (Objects)

Operator Overloading:

- Operator +: **__add__(self, other)**
- Operator -: **__sub__(self, other)**
- Operator *: **__mul__ (self, other)**
- Operator ==: **__eq__(self, other)**
- Operator >: **__gt__(self, other)**
- Operator >=: **__ge__(self, other)**

Others: ***https://www.geeksforgeeks.org/operator-overloading-in-python/***

Classes and Instances (Objects

Operator Overloading:

- Operator +: **__add__(self, other)**
- Operator -: **__sub__(self, other)**
- Operator *: **__mul__ (self, other)**
- Operator ==: **__eq__(self, other)**
- Operator >: **__gt__(self, other)**
- Operator >=: **__ge__(self, other)**

Others: ***https://www.geeksforgeeks.org/operator-overloading-in-python/***

```python
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def __str__(self):   # overload the print method
        return f'2D point:\n \
        --> x-coord = {self.__x_coord}\n \
        --> y-coord = {self.__y_coord}'

    def __add__(self, other):    # overload operator +
        x_coord = self.__x_coord + other.__x_coord
        y_coord = self.__y_coord + other.__y_coord
        return Point2D(x_coord, y_coord)

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt1 = Point2D(3.0, 5.0)
pt2 = Point2D(-2.0, 5.0)
pt = pt1 + pt2
pt.print()
```

Classes and Instances (Objects

Operator Overloading:

- Operator +: **__add__(self, other)**
- Operator -: **__sub__(self, other)**
- Operator *: **__mul__ (self, other)**
- Operator ==: **__eq__(self, other)**
- Operator >: **__gt__(self, other)**
- Operator >=: **__ge__(self, other)**

Others: ***https://www.geeksforgeeks.org/operator-overloading-in-python/***

```
2D point:
  --> x-coord = 1.0
  --> y-coord = 10.0
```

```python
class Point2D:
    # private attributes
    __x_coord = 0.0
    __y_coord = 0.0

    def __init__(self, x_coord, y_coord):
        self.__x_coord = x_coord
        self.__y_coord = y_coord

    def __str__(self):   # overload the print method
        return f'2D point:\n \
        --> x-coord = {self.__x_coord}\n \
        --> y-coord = {self.__y_coord}'

    def __add__(self, other):    # overload operator +
        x_coord = self.__x_coord + other.__x_coord
        y_coord = self.__y_coord + other.__y_coord
        return Point2D(x_coord, y_coord)

    def print(self):
        print('2D point:')
        print(f'  --> x-coord = {self.__x_coord}')
        print(f'  --> y-coord = {self.__y_coord}')

pt1 = Point2D(3.0, 5.0)
pt2 = Point2D(-2.0, 5.0)
pt = pt1 + pt2
pt.print()
```

Classes and Instances (Objects)

Operator Overloading:

- Operator +: **__add__(self, other)**
- Operator -: **__sub__(self, other)**
- Operator *: **__mul__ (self, other)**
- Operator ==: **__eq__(self, other)**
- Operator >: **__gt__(self, other)**
- Operator >=: **__ge__(self, other)**

Others: ***https://www.geeksforgeeks.org/operator-overloading-in-python/***

```
def __eq__(self, other):
    if self.__x_coord == other.__x_coord \
    and self.__y_coord == other.__y_coord:
        return True
    else:
        return False
```

```
pt1 = Point2D(3.0, 5.0)
pt2 = Point2D(-2.0, 5.0)
if pt1 == pt2:
    print('Same points')
else:
    print('Different points')
```

Classes and Instances (Objects)

Inheritance and Polymorphism:

- Inheritance: **class Derived(Base)**
- Polymorphism: **re-definition of a method in a derived class**

Classes and Instances (Objects)

Inheritance and Polymorphism:

```
class Polygon:
    __area__ = 0.0

    def computeArea(self):
        pass

    def print(self):
        print(f'This is a POLYGON')

    def printArea(self):
        print(f'area = {self.__area__}')
```

Classes and Instances (Objects)

Inheritance and Polymorphism:

```python
class Polygon:
    __area__ = 0.0

    def computeArea(self):
        pass

    def print(self):
        print(f'This is a POLYGON')

    def printArea(self):
        print(f'area = {self.__area__}')
```

```python
class Square(Polygon):
    __side__ = 0.0

    def __init__(self, side):
        self.__side__ = side

    def print(self):
        print(f'This is a SQUARE')

    def computeArea(self):
        self.__area__ = self.__side__ * self.__side__
        return self.__area__
```

Classes and Instances (Objects)

Inheritance and Polymorphism:

```python
class Polygon:
    __area__ = 0.0

    def computeArea(self):
        pass

    def print(self):
        print(f'This is a POLYGON')

    def printArea(self):
        print(f'area = {self.__area__}')
```

```python
class Square(Polygon):
    __side__ = 0.0

    def __init__(self, side):
        self.__side__ = side

    def print(self):
        print(f'This is a SQUARE')

    def computeArea(self):
        self.__area__ = self.__side__ * self.__side__
        return self.__area__
```

```python
class Triangle(Polygon):
    __height__ = 0.0
    __base__ = 0.0

    def __init__(self, height, base):
        self.__height__ = height
        self.__base__ = base

    def print(self):
        print(f'This is a TRIANGLE')

    def computeArea(self):
        self.__area__ = 0.5 * self.__height__ * self.__base__
        return self.__area__
```

Classes and Instances (Objects)

Inheritance and Polymorphism:

```python
class Polygon:
    __area__ = 0.0

    def computeArea(self):
        pass

    def print(self):
        print(f'This is a POLYGON')

    def printArea(self):
        print(f'area = {self.__area__}')
```

```python
class Square(Polygon):
    __side__ = 0.0

    def __init__(self, side):
        self.__side__ = side

    def print(self):
        print(f'This is a SQUARE')

    def computeArea(self):
        self.__area__ = self.__side__ * self.__side__
        return self.__area__
```

```python
class Triangle(Polygon):
    __height__ = 0.0
    __base__ = 0.0

    def __init__(self, height, base):
        self.__height__ = height
        self.__base__ = base

    def print(self):
        print(f'This is a TRIANGLE')

    def computeArea(self):
        self.__area__ = 0.5 * self.__height__ * self.__base__
        return self.__area__
```

```python
t = Triangle(3.0, 5.0)
s = Square(4.0)
shape = [t, s]
totalArea = 0.0
for s in shape:
    s.print()
    totalArea = totalArea + s.computeArea()

print(f'Total area = {totalArea}')
```

Classes and Instances (Objects)

Inheritance and Polymorphism:

```python
class Polygon:
    __area__ = 0.0

    def computeArea(self):
        pass

    def print(self):
        print(f'This is a POLYGON')

    def printArea(self):
        print(f'area = {self.__area__}')
```

```python
class Square(Polygon):
    __side__ = 0.0

    def __init__(self, side):
        self.__side__ = side

    def print(self):
        print(f'This is a SQUARE')

    def computeArea(self):
        self.__area__ = self.__side__ * self.__side__
        return self.__area__
```

```python
class Triangle(Polygon):
    __height__ = 0.0
    __base__ = 0.0

    def __init__(self, height, base):
        self.__height__ = height
        self.__base__ = base

    def print(self):
        print(f'This is a TRIANGLE')

    def computeArea(self):
        self.__area__ = 0.5 * self.__height__ * self.__base__
        return self.__area__
```

```python
t = Triangle(3.0, 5.0)
s = Square(4.0)
shape = [t, s]
totalArea = 0.0
for s in shape:
    s.print()
    totalArea = totalArea + s.computeArea()

print(f'Total area = {totalArea}')
```

```
This is a TRIANGLE
This is a SQUARE
Total area = 23.5
```

Modules and Packages

- **Module:** a file
- **Package:** a folder/directory
- To load a module: **import**

Modules and Packages

```python
class Polygon:
    __area__ = 0.0

    def computeArea(self):
        pass

    def print(self):
        print(f'This is a POLYGON')

    def printArea(self):
        print(f'area = {self.__area__}')
```

```python
class Square(Polygon):
    __side__ = 0.0

    def __init__(self, side):
        self.__side__ = side

    def print(self):
        print(f'This is a SQUARE')

    def computeArea(self):
        self.__area__ = self.__side__ * self.__side__
        return self.__area__
```

```python
class Triangle(Polygon):
    __height__ = 0.0
    __base__ = 0.0

    def __init__(self, height, base):
        self.__height__ = height
        self.__base__ = base

    def print(self):
        print(f'This is a TRIANGLE')

    def computeArea(self):
        self.__area__ = 0.5 * self.__height__ * self.__base__
        return self.__area__
```

polygon.py

Modules and Packages

polygon.py

testModule.ipynb

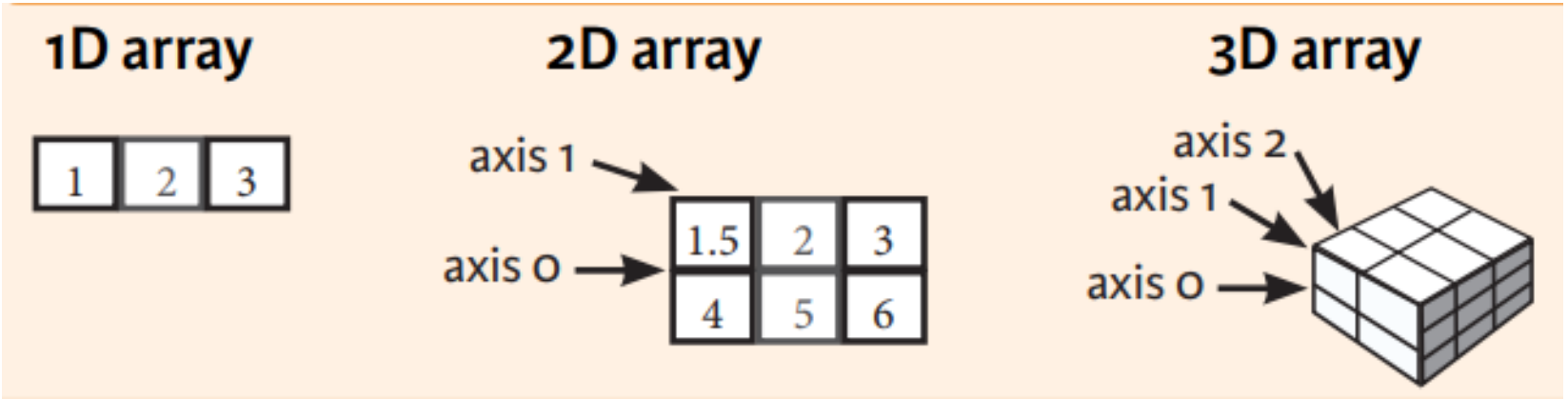
```
import polygon as pl

t = pl.Triangle(3.0, 5.0)
s = pl.Square(4.0)
shape = [t, s]
totalArea = 0.0
for s in shape:
    s.print()
    area = s.computeArea()
    totalArea = totalArea + area
print(f'totalArea = {totalArea}')
```

```
This is a TRIANGLE
This is a SQUARE
totalArea = 23.5
```

Package: NumPy

NumPy

- import numpy as np
- **For handling arrays**



Package: NumPy

Initialization:

- zeros, ones, arange, linspace,

full, eyes, random

- from **lists** or **tuples** with constructor **array()**

Package: NumPy

Initialization:

- zeros, ones, arange, linspace, full, eyes, random
- from **lists** or **tuples** with constructor **array()**

```
[1]: import numpy as np
```

```
[4]: arr1 = np.zeros((2,4))
print(arr1)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
[6]: arr2 = np.arange(10, 25, 5)
print(arr2)
```

```
[10 15 20]
```

```
[8]: arr3 = np.full((3,2), 10, dtype = np.float32)
print(arr3)
```

```
[[10. 10.]
 [10. 10.]
 [10. 10.]]
```

Package: NumPy

Initialization:

- zeros, ones, arange, linspace, full, eyes, random
- from **lists** or **tuples** with constructor **array()**

```
list1 = [[1,2,3], [4,5,6], [7,8,9]]
arr4 = np.array(list1)
print(arr4)
```

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

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5
print(arr6)
```

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

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5
print(arr6)
```

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

```
arr6[0][0] = 1000
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5
print(arr6)
```

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

```
arr6[0][0] = 1000
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[1000    2    3]
 [   4    5    6]
 [   7    8    9]]
arr5 =  [[1000    2    3]
 [   4    5    6]
 [   7    8    9]]
```

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5
print(arr6)
```

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

```
arr6[0][0] = 1000
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

arr5 and arr6 point to the same mem location

```
arr6 =  [[1000    2    3]
 [   4    5    6]
 [   7    8    9]]
arr5 =  [[1000    2    3]
 [   4    5    6]
 [   7    8    9]]
```

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5[0:2][0:2]
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[1 2 3]
 [4 5 6]]
arr5 =  [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5[0:2][0:2]
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[1 2 3]
 [4 5 6]]
arr5 =  [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
arr6[:][:] = 100
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5[0:2][0:2]
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[1 2 3]
 [4 5 6]]
arr5 =  [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
arr6[:][:] = 100
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[100 100 100]
 [100 100 100]]
arr5 =  [[100 100 100]
 [100 100 100]
 [  7   8   9]]
```

Package: NumPy

Slicing/Viewing:

- Names as **pointers**

arr5 and arr6 point to the same mem location

arr6 is a *view window* of arr5

→ Memory efficient

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr6 = arr5[0:2][0:2]
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[1 2 3]
 [4 5 6]]
arr5 =  [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
arr6[:][:] = 100
print('arr6 = ', arr6)
print('arr5 = ', arr5)
```

```
arr6 =  [[100 100 100]
 [100 100 100]]
arr5 =  [[100 100 100]
 [100 100 100]
 [  7   8   9]]
```

Package: NumPy

Slicing/Viewing:

- Want to have **separate** arrays?

Package: NumPy

Slicing/Viewing:

- Want to have **separate** arrays?
- **np.array()** constructor
- **np.copy()** constructor

Package: NumPy

Slicing/Viewing:

- Want to have **separate** arrays?
- **np.array()** constructor
- **np.copy()** constructor

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

Package: NumPy

Slicing/Viewing:

- Want to have **separate** arrays?
- **np.array()** constructor
- **np.copy()** constructor

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr7 = np.array(arr5[0:2][0:2])
print('arr7 = ', arr7)
```

```
arr7 =  [[100 100 100]
 [100 100 100]]
```

```
arr7[:][:]=1
print('arr7 = ', arr7)
print('arr5 = ', arr5)
```

```
arr7 =  [[1 1 1]
 [1 1 1]]
arr5 =  [[100 100 100]
 [100 100 100]
 [  7   8   9]]
```

Package: NumPy

Slicing/Viewing:

- Want to have **separate** arrays?
- **np.array()** constructor
- **np.copy()** constructor

```
tuple1 = ((1,2,3), (4, 5, 6), (7, 8, 9))
arr5 = np.array(tuple1)
print(arr5)
```

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

```
arr8 = np.copy(arr5[0:2][0:2])
print('arr8 = ', arr8)
```

```
arr8 =  [[100 100 100]
 [100 100 100]]
```

```
arr8[:][:]=10
print('arr8 = ', arr8)
print('arr5 = ', arr5)
```

```
arr8 =  [[10 10 10]
 [10 10 10]]
arr5 =  [[100 100 100]
 [100 100 100]
 [  7   8   9]]
```

Package: NumPy

Operations:

- **Arithmetic:** +, -, *, /
- **Comparison:** ==, >, <
- **Aggregation:** sum(), min(), max(), cumsum(), mean(), median()
- **Manipulation:** reshape(), sort(), transpose(), concatenate(), resize(), append(), insert(), delete()

```
arr1 = np.array([[1, 2, 3, 4, 5, 6]])
arr2 = np.full((1,6), 10)
arr3 = arr1 + arr2
print('arr3 = ', arr3)
arr4 = arr1.reshape((2, 3))
print('arr4 = ', arr4)
```

```
arr3 =  [[11 12 13 14 15 16]]
arr4 =  [[1 2 3]
 [4 5 6]]
```

Package: SciPy

SciPy

- import numpy as np
- From scipy import linalg, sparse
- **Linear algebra operations and solvers**
- **Sparse matrices**

Package: SciPy

SciPy

```
A = np.matrix(np.random.random((2,2)))
B = np.mat(np.random.random((2,5)))
C = np.mat([[3,4], [5,6]])
print('A = ', A)
print('B = ', B)
print('C = ', C)
```

```
A =  [[0.42369401 0.97756087]
 [0.4997074  0.58191561]]
B =  [[0.78538671 0.85345464 0.3900455  0.0892106  0.03118969]
 [0.62837133 0.3052462  0.30408515 0.08103062 0.52292209]]
C =  [[3 4]
 [5 6]]
```

Package: SciPy

SciPy

```
A.I
```

```
matrix([[-2.40520382,  4.0405054 ],
        [ 2.0654166 , -1.7512341 ]])
```

```
A.T
```

```
matrix([[0.42369401, 0.4997074 ],
        [0.97756087, 0.58191561]])
```

```
linalg.det(A)
```

```
-0.24194024691715824
```

```
b = np.mat(np.random.random((2,1)))
linalg.solve(A,b)
```

```
array([[-1.39276882],
       [ 1.38985265]])
```

Package: Matplotlib

Matplotlib

- import matplotlib.pyplot as plt
- **2D data visualization**

Package: Matplotlib

Matplotlib

- import matplotlib.pyplot as plt
- **2D data visualization**

```
import numpy as np
import matplotlib.pyplot as plt
```

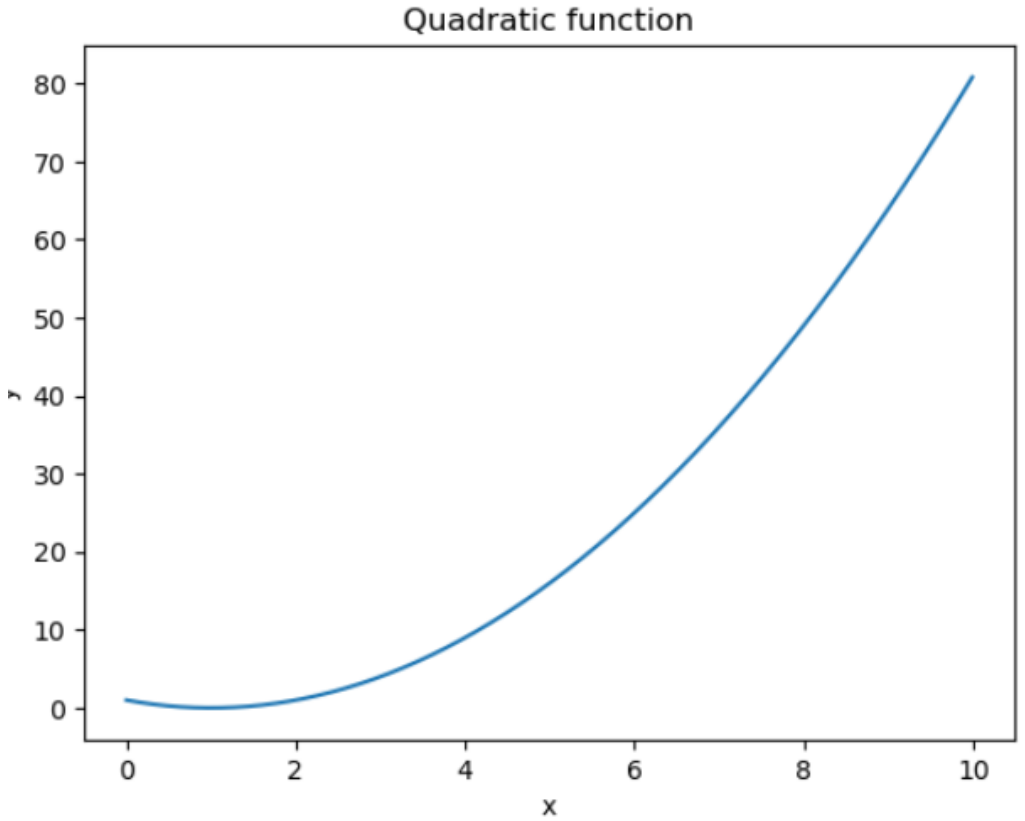
```
x = np.arange(0.0, 10.0, 0.01)
y = x*x - 2*x + 1
```

```
fig = plt.figure()
plt.plot(x, y)
plt.title('Quadratic function')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('figure.png')
```

Package: Matplotlib

Matplotlib

- import matplotlib.pyplot as plt
- **2D data visualization**




```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.arange(0.0, 10.0, 0.01)
y = x*x - 2*x + 1
```

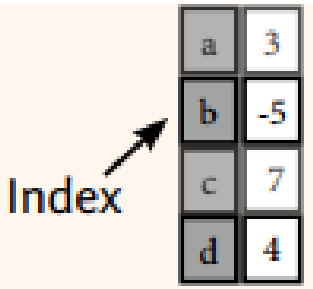
```
fig = plt.figure()
plt.plot(x, y)
plt.title('Quadratic function')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('figure.png')
```

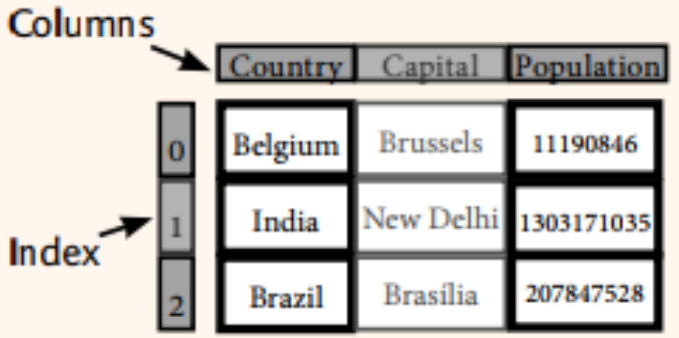
Package: Pandas

Pandas

- import pandas as pd
- **1D series and 2D dataframes**

A **one-dimensional** labeled array capable of holding any data type





A **two-dimensional** labeled data structure with columns of potentially different types

Source: Python for Data Science cheat sheet *(www.datacamp.com)*

References

1. Geeksforgeeks.org – python
2. M. Lutz, *“Learning Python”*, 4th. Ed., O’Reilly
3. F. Fletcher, D. Amos, D. Bader, J. Jablonski, *“Python Basics: A Practical Introduction to Python 3”*, Real Python, realpython.com

Thank you for your attention!