Western

UNIVERSITY · CANADA

AI-in-Action Heroes

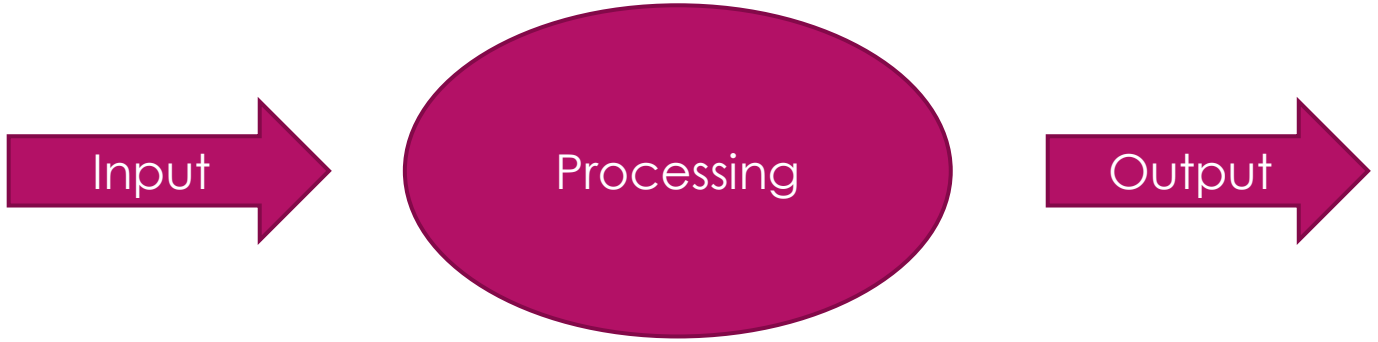
SUMMER ACADEMY COURSE OFFERED BY AISE PROGRAM

Object-Oriented Features in Python

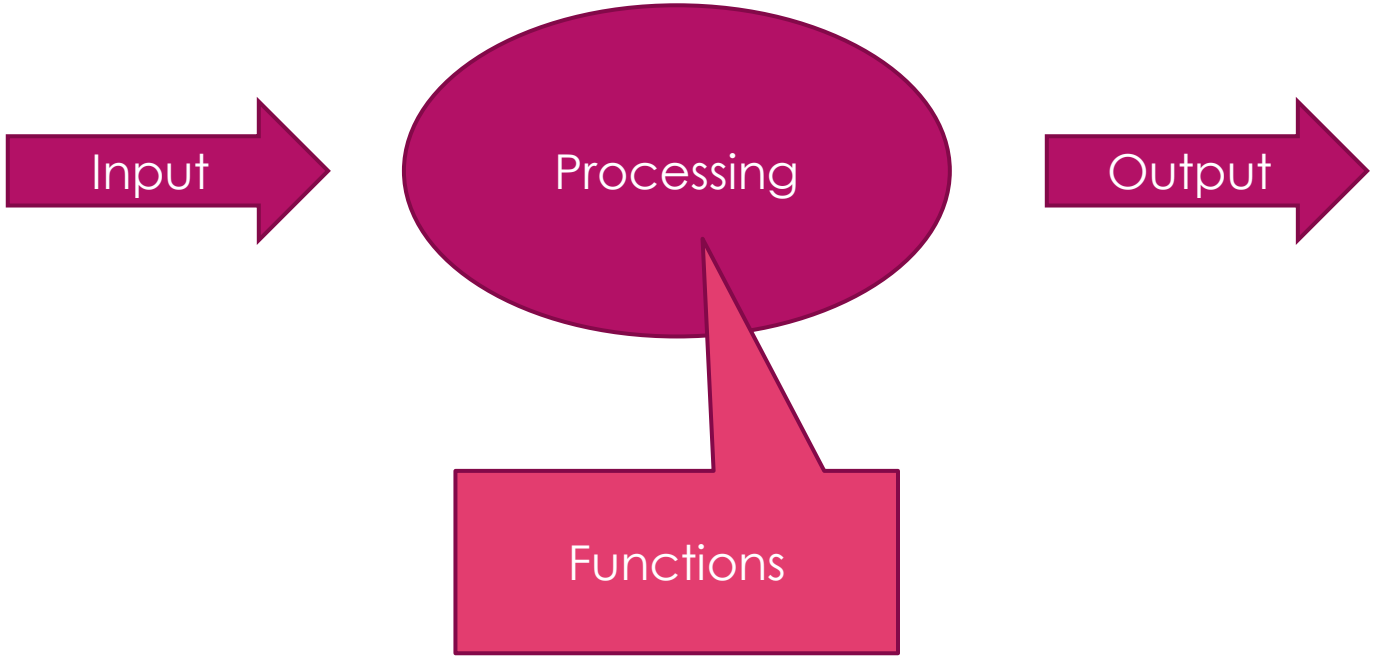
Object-Oriented Vs. Structured Paradigms

A PROGRAMMING PARADIGM DETERMINES THE WAY THE DATA AND THE PROCESSES ARE ORGANIZED IN THE PROGRAM.

Structured Programing



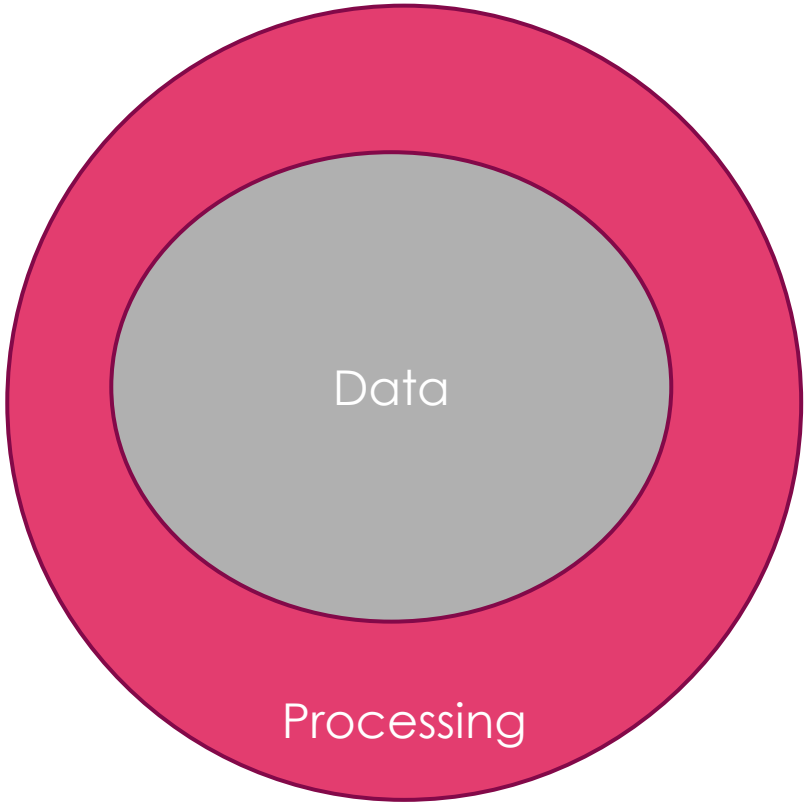
Structured Programing



Object-Oriented Paradigm

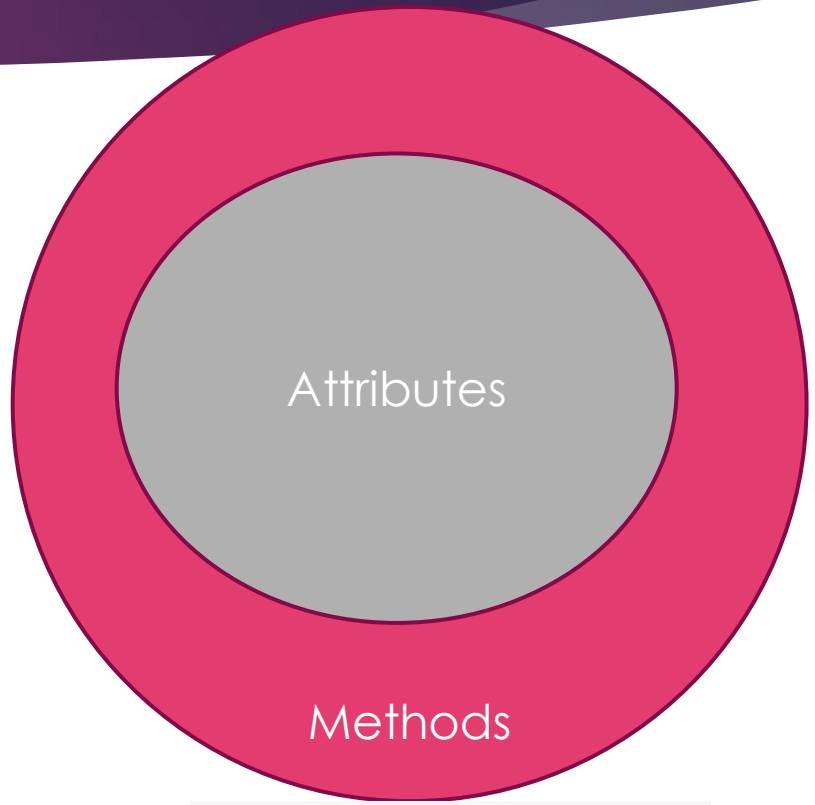
Object-Oriented Programming

- Group the data and processes operating on them in one bundle



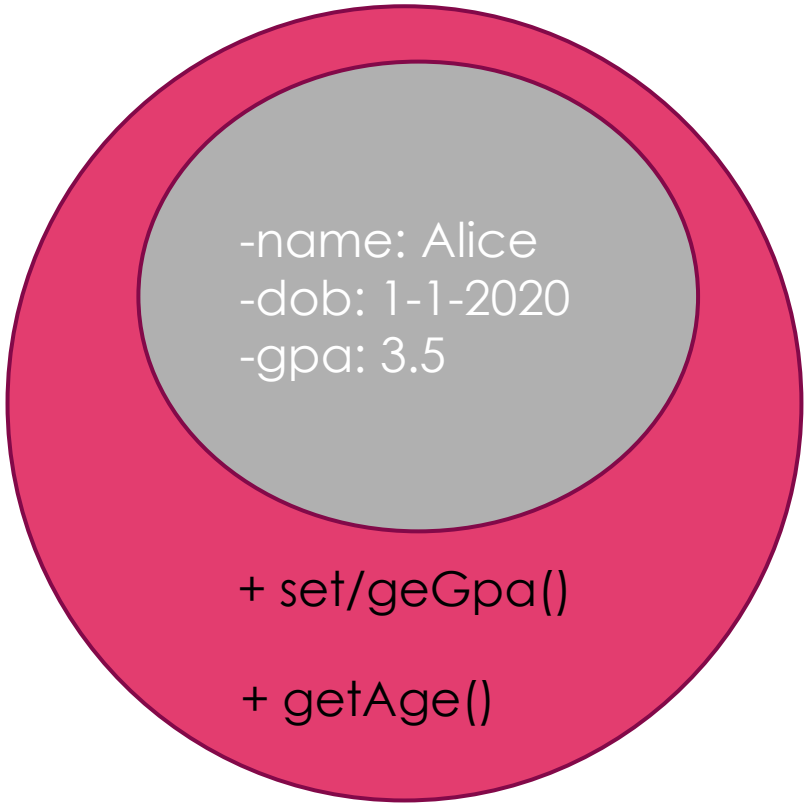
Object-Oriented Programming

- Group the data and processes operating on them in one bundle.
- That bundle is called object
- The data are called **attributes / properties**
- The processing implemented in "**methods**"



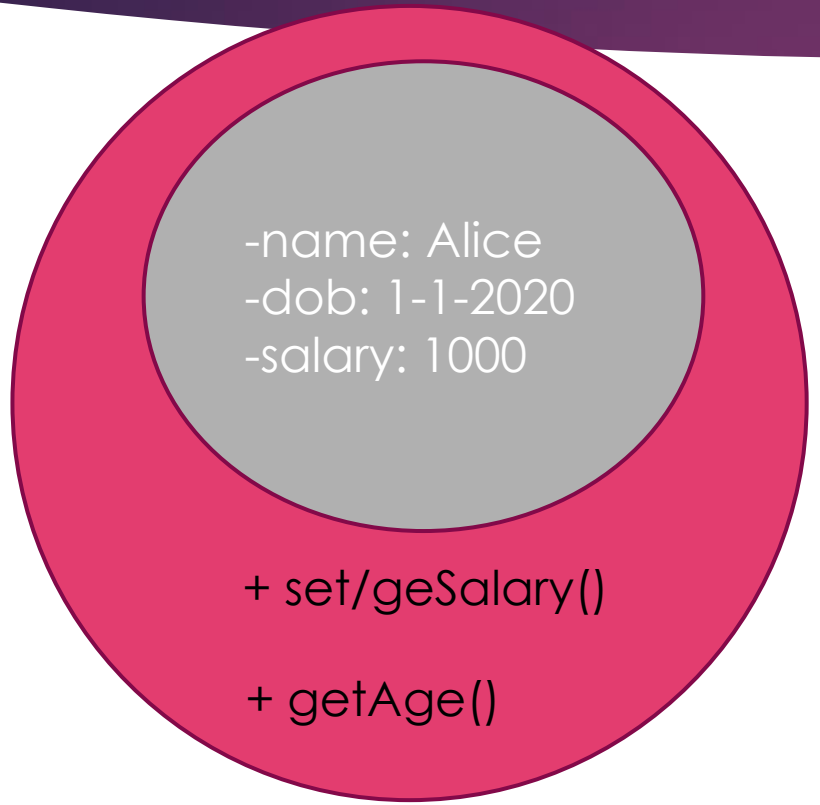
[OOP]Object

Object-Oriented Programming

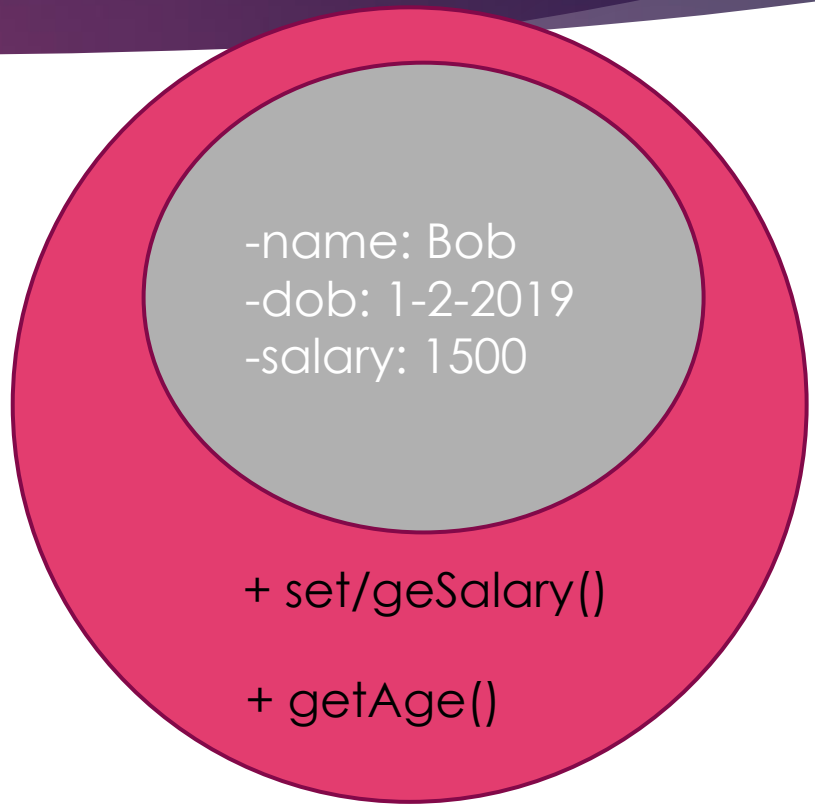


Object :o1

Object-Oriented Programming

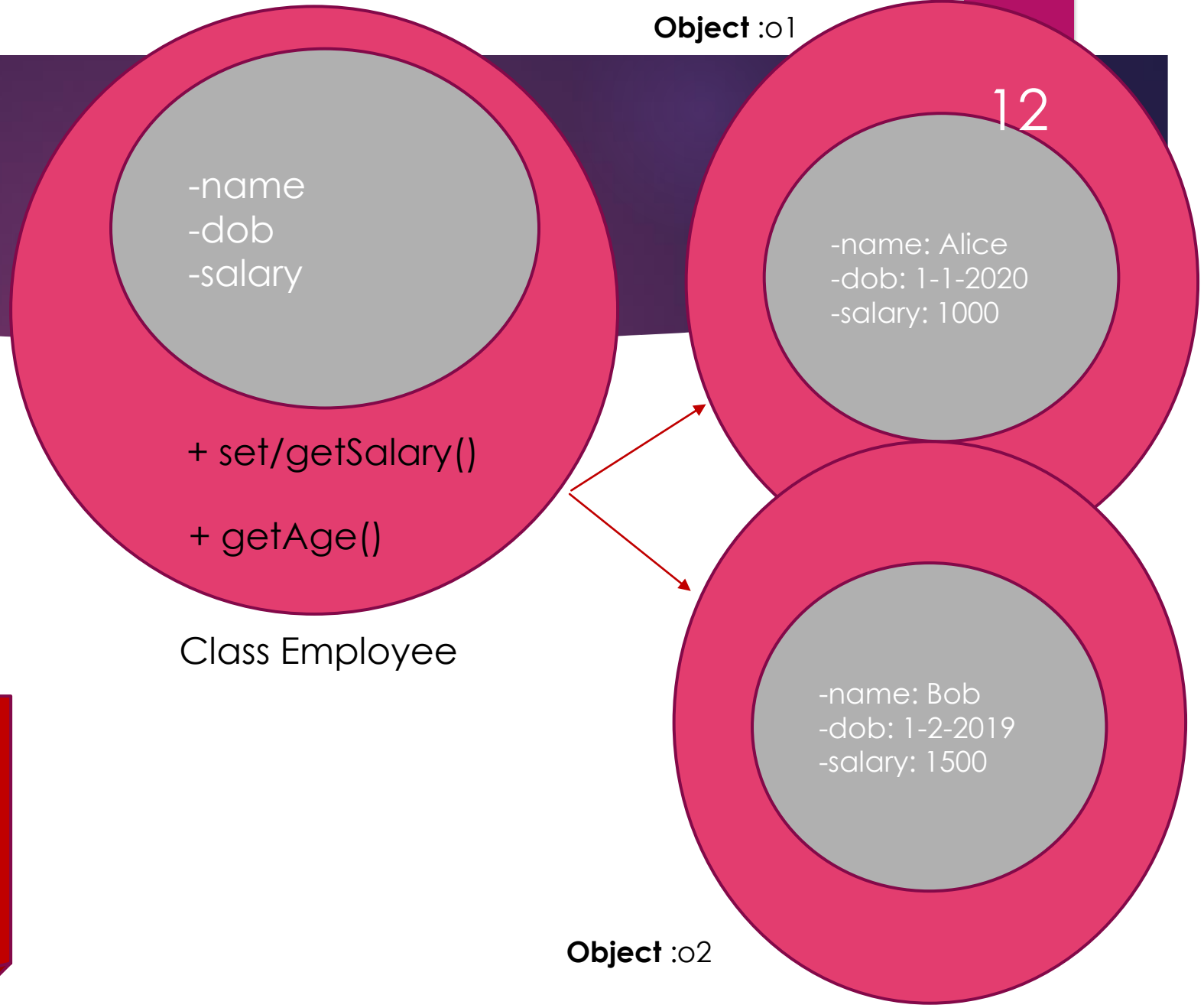


Object :o1



Object :o2

Object-Oriented Programming



- **[OOP] Class**
  - A template for similar objects
- **[OOP] Instantiation**
  - Creating an instance of a Class

Class definition & Objects instantiation

```python
class Employee:
    name = ''
    email = ''

emp1 = Employee()
emp1.name = "John"
emp1.email = 'john@gmail.com'
emp1.salary = 1000

print(emp1.email, emp1.name, emp1.salary)

emp2 = Employee()
emp2.name = "Smith"
emp2.email = 'smith@gmail.com'
del emp2.email

print(emp2.email, emp2.name, emp2.salary)
```

Class definition & Objects instantiation

```python
class Employee:
    name = ''
    email = ''

emp1 = Employee()
emp1.name = "John"
emp1.email = 'john@gmail.com'
emp1.salary = 1000

print(emp1.email, emp1.name, emp1.salary)

emp2 = Employee()
emp2.name = "Smith"
emp2.email = 'smith@gmail.com'
del emp2.email

print(emp2.email, emp2.name, emp2.salary)
```

A new property is added to emp1 object

emp2 doesn't have this property so this will give an error

Class definition & Objects instantiation

```python
class Employee:
    name = ''
    email = ''

emp1 = Employee()
emp1.name = "John"
emp1.email = 'john@gmail.com'
emp1.salary = 1000

print(emp1.email, emp1.name, emp1.salary)

emp2 = Employee()
emp2.name = "Smith"
emp2.email = 'smith@gmail.com'
del emp2.email

print(emp2.email, emp2.name, emp2.salary)
```

Sets the value back to the initial value

This doesn't cause an error

The __init__ built-in method

- The built-in __init__ method is called automatically every time we create an instance of the class.
- It has to have at least one parameter which is a reference to the object being created
- 'self' is just a name not a keyword, we can give it any other name
  - It containts the reference to the object

```
class Employee:
    name = ''
    email = ''

    def __init__(self) -> None:
        pass

emp1 = Employee()
emp1.name = 'John'
emp1.email = 'john@gmail.com'
emp1.salary = 1000

print(emp1.email, emp1.name, emp1.salary)
```

Defining Methods

- When defining methods in the class we need to define the 'self' parameter even if it's not needed in the body of the function

```python
class Employee:
    def __init__(self, name, email, salary):
        self.name = name
        self.email = email
        self.salary = salary

    def printEmpInfo(self):
        print(self.name, self.email,self.salary)

    def anotherMethod():
        print("Doesn't need to access the object's attributes")

emp1 = Employee('John', 'john@gmail.com', 1000 )
emp1.printEmpInfo()
emp1.anotherMethod()
```

```
Exception has occurred: TypeError ×
anotherMethod() takes 0 positional arguments but 1 was given
  File "C:\Users\shaim\OneDrive - The University of Western Ontario\ECE\SE
line 15, in <module>
```

Python Modules & Libraries

Python Modules

- In Python a file is considered a module
- The name of the file (without the .py extension) is the name of the module
- Modules can be imported into other modules using the keyword ***'import'***

EXPLORER

OPEN EDITORS

PYTHONMODULES
- __pycache__
- calculator.py
- main.py

calculator.py > Person > __init__ > email

```python
PI = 3.14
x = 10

def add(a,b):
    c = a + b
    return c

def subtract(a,b):
    c = a - b
    return c

class Person:
    def __init__(self,name, email) -> None:
        self.name = name
        self.email = email
```

main.py > ...

```python
import calculator

print(calculator.PI) # => 3.14

result = calculator.add(5,8)
print (result) # => 13

person1 = calculator.Person("Alice","alice@gmail.com")
print(person1.name)

print(calculator.x) # => 10
calculator.x += 5
print(calculator.x) # => 15
```

Python Modules

- In Python a file is considered a module
- The name of the file (without the .py extension) is the name of the module
- Modules can be imported into other modules using the keyword ***'import'***

By importing a module we can have access to its content using the dot (.) operator following its name

EXPLORER

OPEN EDITORS

PYTHONMODULES
- __pycache__
- calculator.py
- main.py

calculator.py > Person > __init__ > email

```python
PI = 3.14
x = 10

def add(a,b):
    c = a + b
    return c

def subtract(a,b):
    c = a - b
    return c

class Person:
    def __init__(self,name, email) -> None:
        self.name = name
        self.email = email
```

main.py > ...

```python
import calculator

print(calculator.PI) # => 3.14

result = calculator.add(5,8)
print (result) # => 13

person1 = calculator.Person("Alice","alice@gmail.com")
print(person1.name)

print(calculator.x) # => 10
calculator.x += 5
print(calculator.x) # => 15
```

Python Modules

- In Python a file is considered a module
- The name of the file (without the .py extension) is the name of the module
- Modules can be imported into other modules using the keyword ***'import'***

Using the 'as' keyword we can define an alias for the module for convenience

EXPLORER

OPEN EDITORS

PYTHONMODULES

- __pycache__
- calculator.py
- main.py

calculator.py > Person > __init__ > email

```python
PI = 3.14
x = 10

def add(a,b):
    c = a + b
    return c

def subtract(a,b):
    c = a - b
    return c

class Person:
    def __init__(self,name, email) -> None:
        self.name = name
        self.email = email
```

main.py > ...

```python
import calculator as calc

print(calc.PI) # => 3.14

result = calc.add(5,8)
print (result) # => 13

person1 = calc.Person("Alice","alice@gmail.com")
print(person1.name)

print(calc.x) # => 10
calc.x += 5
print(calc.x) # => 15
```

Python Modules

- In Python a file is considered a module
- The name of the file (without the .py extension) is the name of the module
- Modules can be imported into other modules using the keyword ***'import'***

We can import one (or more) element from the module and use there names directly without the dot (.) operator

EXPLORER

OPEN EDITORS

PYTHONMODULES
- __pycache__
- calculator.py
- main.py
- payroll.py

calculator.py > Person > __init__ > email

```python
PI = 3.14
x = 10

def add(a,b):
    c = a + b
    return c

def subtract(a,b):
    c = a - b
    return c

class Person:
    def __init__(self,name, email) -> None:
        self.name = name
        self.email = email
```

main.py | payroll.py

payroll.py > ...

```python
from calculator import Person

person1 = Person('Alice', 'alice@gmail.com')
print(person1.name)

class Employee(Person):
    def __init__(self,name, email,salary) -> None:
        super().__init__(name,email)
        self.salary = salary

emp = Employee('Bob','bob@gmail.com',1000)
print(emp.name, emp.salary)
```

Let's use another editor for this exercise