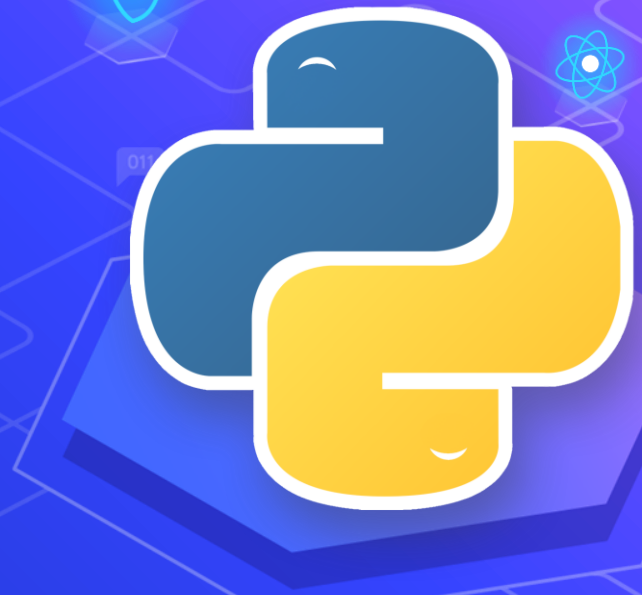
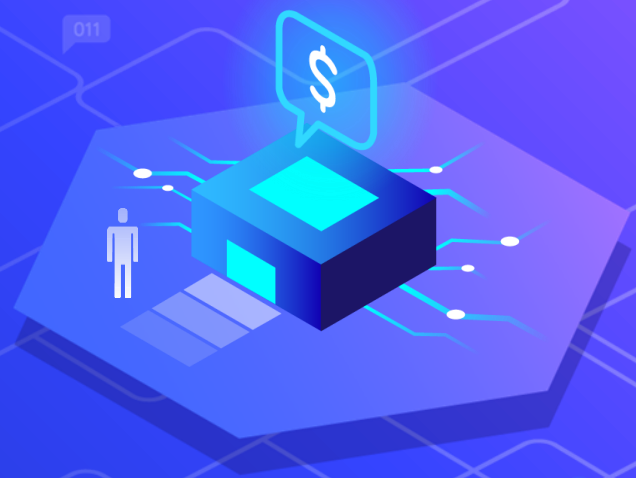
Object Oriented Programming (OOP)



Why OOP ?!

Let’s see Why ^_^

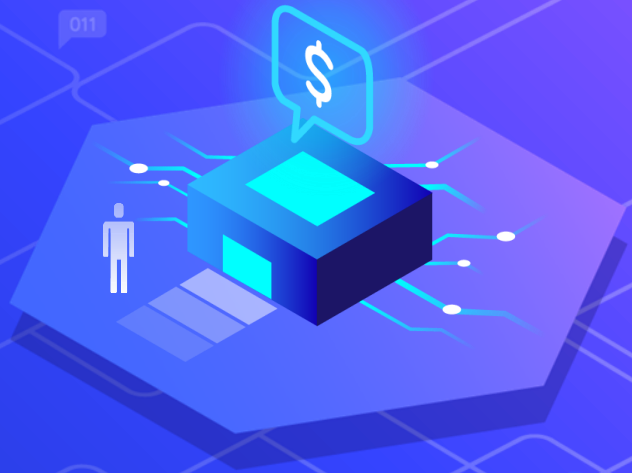


Calculate Sum 10000 times ?!

```python
x = 10
y = 20
print(x + y)

x = 50
y = 70
print(x + y)

x = 90
y = 200
print(x + y)
```

Calculate Sum 10000 times ?!

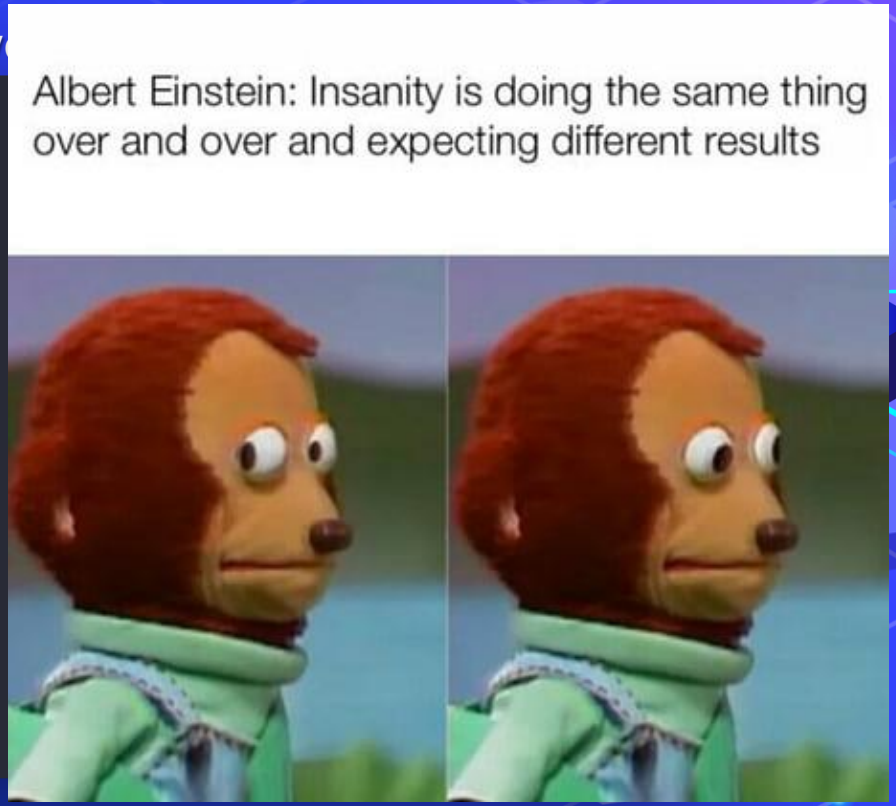
What if I want to modify code to multiply ?



Calculate Sum 10000 times ?!

What if I w

```python
x = 10
y = 20
print(x + y)

x = 50
y = 70
print(x + y)

x = 90
y = 200
print(x + y)
```



Calculate Sum 10000 times ?!

What if I want to modify code to multiply ?

```python
def calculate(x, y):
  return x + y # to multiply just make '+' -> '*'

calculate(1, 2) # 3
calculate(10, 20) # 30
calculate(50, 40) # 90
...
```

Make Full Calculator ?

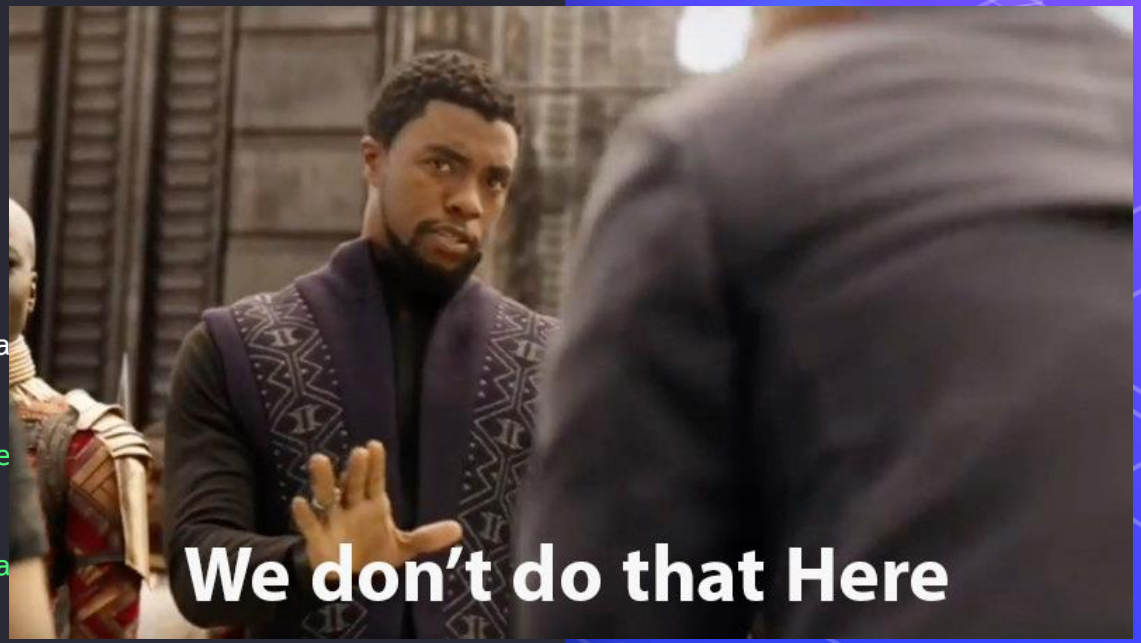
```python
def summ(x, y):
  return x + y

def multiply(x, y):
  return x * y

def subtract(x, y):
  return x - y

def divide(x, y):
  return x / y

summ(1, 2) # 3
multiply(10, 20) # 200
subtract(50, 40) # 10
divide(50, 2) # 25
...
```

Make Lots of Features ?

```python
def summ(x, y):
  pass

def subtract(x, y):
  pass

def read_file(path):
  pass

def write_data_in_file(data, path):
  pass

def get_data_from_internet(url):
  pass

def delete_data_from_database(data):
  pass

...
```

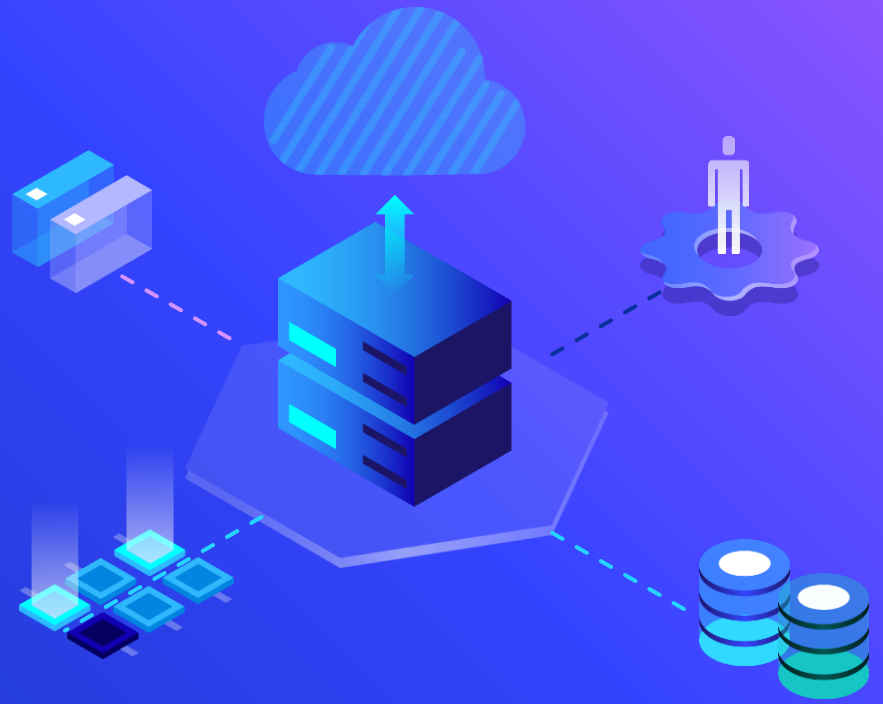
Make Lots of Features ?

```
def summ(x, y):
  pass

def subtract(x, y):
  pass

def read_file(path):
  pass

def write_data_in_file(da
  pass

def get_data_from_interne
  pass

def delete_data_from_data
  pass

...
```



What is OOP ?!

Everything is an **Object**



Dog

Attributes

- Size
- Color
- Breed
- Age
- ...

Methods

- Run()
- Park()
- Eat()
- Sleep()
- ...

Car

Attributes

- Model
- Color
- Plate Number
- Speed
- ...

Methods

- Steer()
- Back()
- Break()
- Throttle()
- ...

Person

Attributes

- Name
- Age
- Address
- Gender
- ...

Methods

- Eat()
- Sleep()
- Walk()
- Pray()
- ...

Camera

Attributes

- Lens_width
- Has_Flash
- Depth
- ...

Methods

- Take_Photo()
- Take_Video()
- Toggle_Flash()
- Zoom()
- ...

Classes & Objects

Person

Attributes

- Name
- Age
- Address
- Gender
- ...

Methods

- Eat()
- Sleep()
- Walk()
- Pray()
- ...

Class: Person

Object 1

- Name: Eslam
- Age: 26
- Gender: Male

- Eat Meat
- Sleep at 9 pm

Object 2

- Name: Ahmed
- Age: 15
- Gender: Male

- Eat Chicken
- Sleep at 12 am

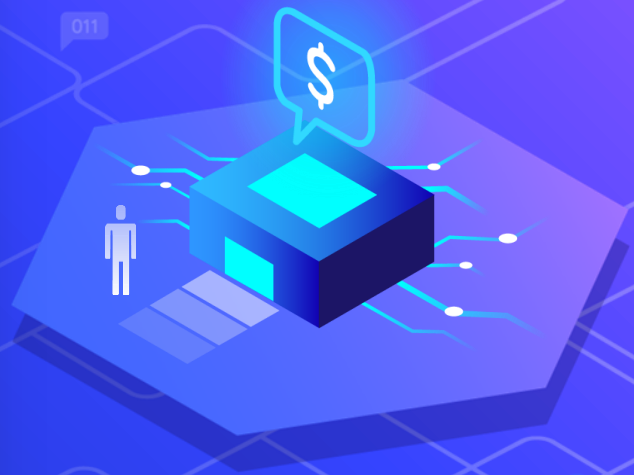
Object 3

- Name: Sara
- Age: 30
- Gender: Female

- Eat Fish
- Sleep at 6 am

OOP in Python ?

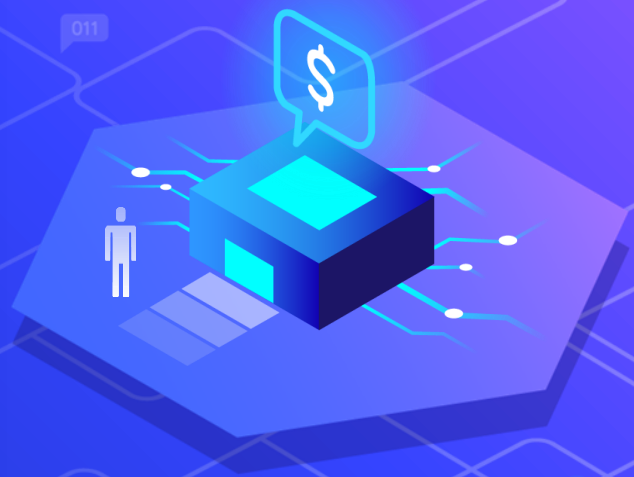
```python
class Circle:
    pi = 3.14

    # Circle gets instantiated with a radius (default is 1)
    def __init__(self, radius=1):
        self.radius = radius


    # Method for resetting Radius
    def getArea(self):
        return (self.radius ** 2) * self.pi


    # Method for getting Circumferenc
    def getCircumference(self):
        return self.radius * self.pi * 2


c1 = Circle()
print(c1.radius)                # 1
print(c1.getArea())             # 3.14
print(c1.getCircumference())    # 6.28


c2 = Circle(10)
print(c2.radius)                # 10
print(c2.getArea())             # 314.0
print(c2.getCircumference())    # 62.8
```



Circle

OOP in Python ?

```python
class Person:

    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender


    def greet(self):
        if self.gender == 'male':
          print('Hello, Mr. ' + self.name)
        elif self.gender == 'female':
          print('Hello, Mrs. ' + self.name)


    def is_old(self):
        return (self.age >= 60)


ahmed = Person('ahmed', 20, 'male')
mohammed = Person('mohammed', 67, 'male')
sara = Person('sara', 30, 'female')

ahmed.is_old() # false
mohammed.is_old() # true

ahmed.greet()  # Hello, Mr. ahmed
sara.greet()  # Hello, Mrs. sara

```



Person

OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading
- Static Attributes & Methods
- Class Methods

Data Hiding (Encapsulation)

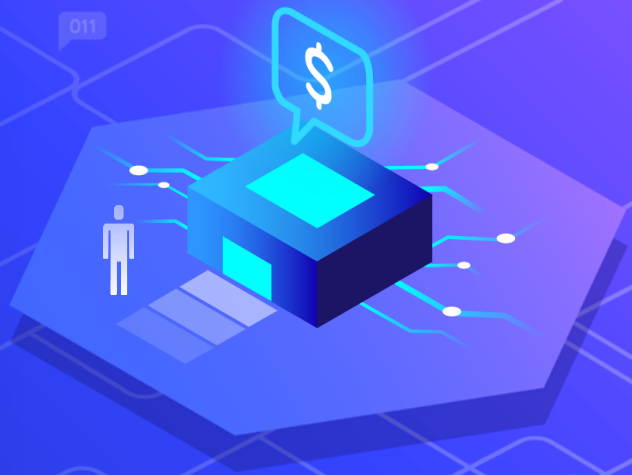
Data Hiding (Encapsulation)

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

Data Hiding (Encapsulation)

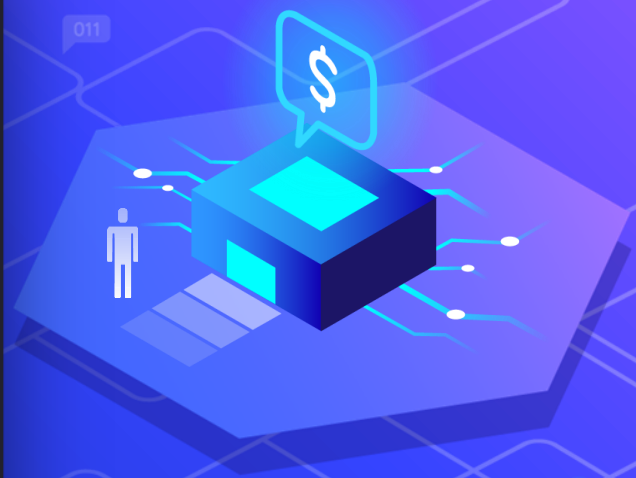
Private members

```python
class Circle:
    pi = 3.14


    # Circle gets instantiated with a radius (default is 1)
    def __init__(self, radius=1):
        self.__radius = radius

    # Method for resetting Radius
    def getArea(self):
        return self.__radius * self.__radius * self.pi

    # Method for getting Circumference
    def getCircumference(self):
        return self.__radius * self.pi * 2



c1 = Circle(10)

print('Radius is: ',c1.__radius)

"""
AttributeError: 'Circle' object has no attribute '__radius'
"""
```



Data Hiding (Encapsulation) Setter and Getter

```python
class Circle:
    pi = 3.14


    # Circle gets instantiated with a radius (default is 1)
    def __init__(self, radius=1):
        if type(radius) == int:
            self.__radius = radius
        else:
            self.__radius = 1


    # Setter
    def set_radius(self, new_radius):
        if type(new_radius) == int:
            self.__radius = new_radius
        else:
            print('this is not an interger')

    # Getter
    def get_radius(self):
        print(f'the radius is: {self.__radius}')
```



OOP Concepts

- Data Hiding (Encapsulation)
- **Inheritance**
- Polymorphism
- Special Functions and Operators Overloading
- Static Attributes & Methods
- Class Methods

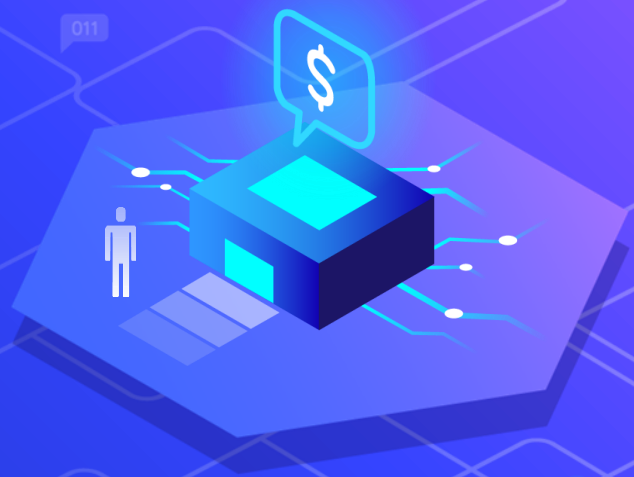
Inheritance

Inheritance

It is a way to form new classes using classes that have already been defined. The newly formed classes are called derived classes, the classes that we derive from are called base classes.

Inheritance

```python
class Animal:

    def __init__(self):
        self.species = 'mammal'
        print("Animal created")

    def whoAmI(self):
        print("Animal")

    def eat(self):
        print("Eating")

# ---------------------------------------------- #

class Dog(Animal):

    def __init__(self):
        Animal.__init__(self)    # call parent __init__
        self.sound = 'High'
        self.love_bones = True
        print("Dog created")

    def bark(self):
        print(f'Woof Woof with {self.sound} Sound')

    def eat(self):
        if self.love_bones:
            print('Love eating bones')
        else:
            print('Love meat')

# ---------------------------------------------- #


sam = Dog()
# Animal created
# Dog created

sam.species     # mammal
sam.love_bones  # True

sam.whoAmI()    # Animal
sam.eat()       # Love eating bones
sam.bark()      # Whao Whao with High Sound
```



OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- **Polymorphism**
- Special Functions and Operators Overloading
- Static Attributes & Methods
- Class Methods

Polymorphism

Polymorphism

polymorphism refers to the way in which different object classes can share the same method name but each acts differently.

Polymorphism

```python
class Dog:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return self.name+' says Woof!'

class Cat:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return self.name+' says Meow!'


jack = Dog('Jack')
meshmesh = Cat('Meshmesh')


for pet in [jack,meshmesh]:
    print(pet.speak())
```

OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- **Special Functions and Operators Overloading**
- Static Attributes & Methods
- Class Methods

Special Functions and Operators Overloading

Math Operators

- `__add__`
- `__sub__`
- `__mul__`
- `__truedivi__`
- `__floordiv__`
- `__mod__`
- `__pow__`

Comparison Operators

- `__eq__`
- `__nq__`
- `__lt__`
- `__le__`
- `__gt__`
- `__ge__`

Other Operators

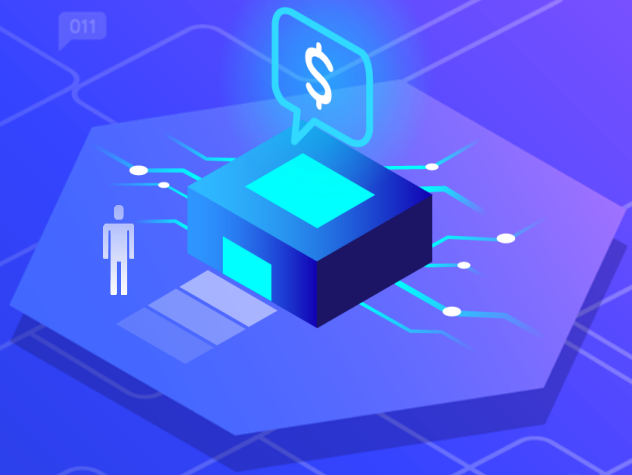
- `__init__`
- `__str__`
- …

OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading
- **Static Attributes & Methods**
- Class Methods

Static Attributes & Methods

```python
class Calculator:

    pi = 3.14

    @staticmethod
    def summ(x, y):
        return x + y

    @staticmethod
    def sub(x, y):
        return x - y

    @staticmethod
    def mul(x, y):
        return x * y

    @staticmethod
    def div(x, y):
        if y == 0:
            return 'Cant divide on zero'
        else:
            return x / y

# ------------------------------------------------ #
Calculator.pi               # 3.14
Calculator.summ(10, 20)     # 30
Calculator.sub(30, 20)      # 10
Calculator.mul(10, 20)      # 200
Calculator.div(100, 20)     # 5
Calculator.div(100, 0)      # Cant divide on zero
```

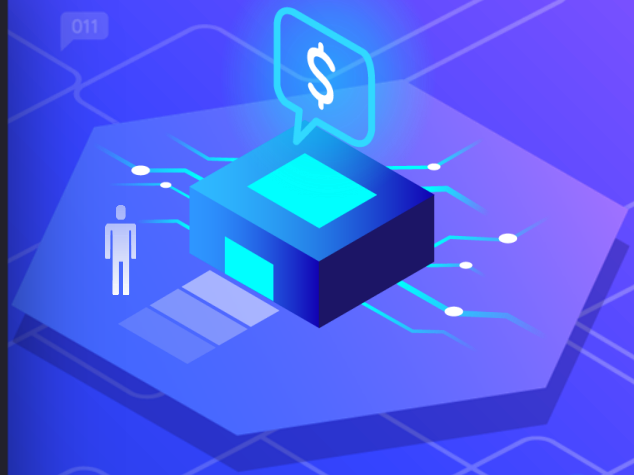


OOP Concepts

- Data Hiding (Encapsulation)
- Inheritance
- Polymorphism
- Special Functions and Operators Overloading
- Static Attributes & Methods
- **Class Methods**

Class Methods

```python
class Apple:

    _counter = 0

    @staticmethod
    def about_apple():
        print('Apple is good for you.')


    @classmethod
    def make_apple_juice(cls, number_of_apples):
        print('Make juice:')
        for i in range(number_of_apples):
            cls._juice_this(i)

    @classmethod
    def _juice_this(cls, apple):
        print(f'Juicing {apple}...')
        cls._counter += 1
```

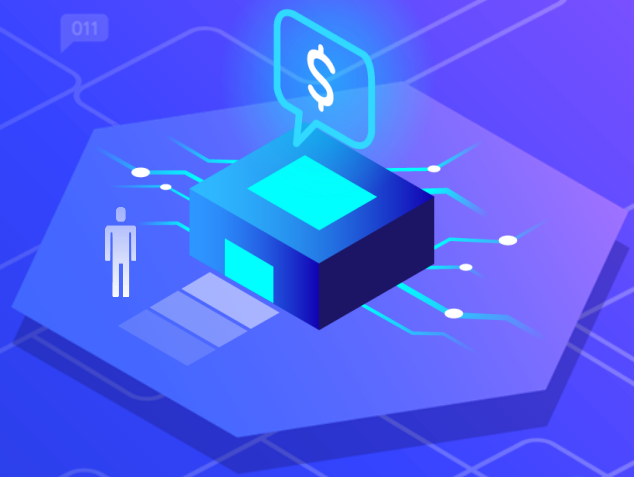


Other Concepts

- Exceptions
- Decorators
- Iterators and Generators

Exceptions

```python
try:
    lst = [1,2,3,4,5]
    print(lst[10])

except ZeroDivisionError:
    print('cant divide on zero')

except IndexError:
    print('you accessed non found index')

except ValueError:
    print('this is value error')

except Exception as e:
    print(type(e).__name__)

else:
    print('No Error')

finally:
    print('Printed if there is exception or no')
```



Check Python Error Types

https://docs.python.org/3/library/exceptions.html

Other Concepts

- Exceptions
- **Decorators**
- Iterators and Generators

Decorators

```python
def new_decorator(func):

    def wrap_func():
        print("Code would be here, before executing the func")
        func()
        print("Code here will execute after the func()")

    return wrap_func

def func_needs_decorator():
    print("This function is in need of a Decorator")


func_needs_decorator()
"""
This function is in need of a Decorator
"""

func_needs_decorator = new_decorator(func_needs_decorator)

func_needs_decorator()
"""
Code would be here, before executing the func
This function is in need of a Decorator
Code here will execute after the func()
"""
```

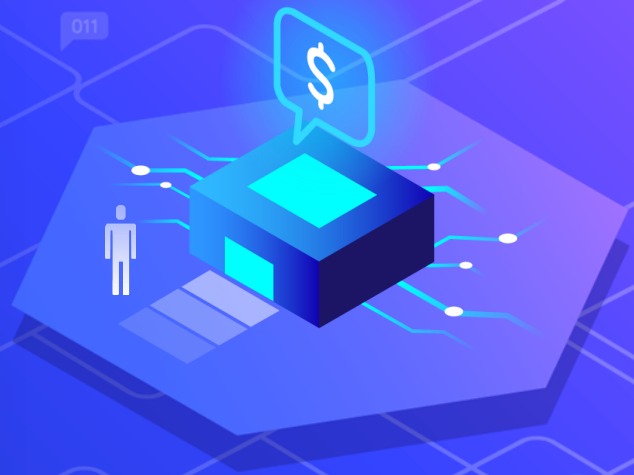
```python
def new_decorator(func):

    def wrap_func():
        print("Code would be here, before executing the func")

        func()

        print("Code here will execute after the func()")

    return wrap_func


@new_decorator
def func_needs_decorator():
    print("This function is in need of a Decorator")


func_needs_decorator()
"""
Code would be here, before executing the func
This function is in need of a Decorator
Code here will execute after the func()
"""
```

Other Concepts

- Exceptions
- Decorators
- **Iterators and Generators**

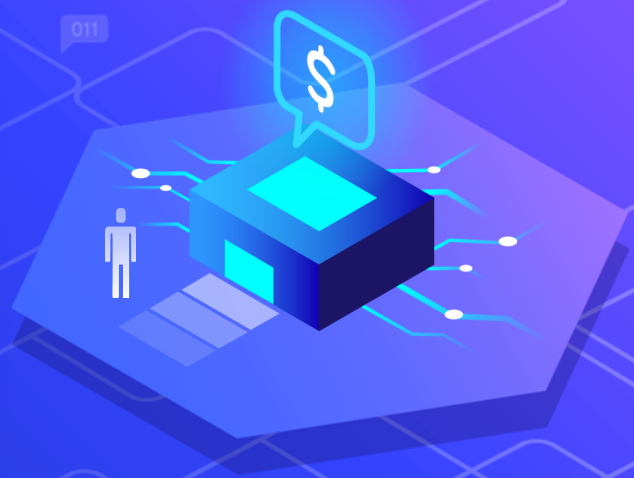
Iterators

```python
# define a list
my_list = [4, 7, 0, 3]

# get an iterator using iter()
my_iter = iter(my_list)

## iterate through it using next()

# prints 4
print(next(my_iter))

# prints 7
print(next(my_iter))

## next(obj) is same as obj.__next__()

# prints 0
print(my_iter.__next__())

# prints 3
print(my_iter.__next__())

## This will raise StopIteration error, no items left
next(my_iter)
```



Custom Iterators

```python
class PowTwo:

    def __init__(self, max = 0):
        self.max = max

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n <= self.max:
            result = 2 ** self.n
            self.n += 1
            return result
        else:
            raise StopIteration


a = iter(PowTwo(4))
a.__next__()   # 1
a.__next__()   # 4
a.__next__()   # 8
a.__next__()   # 16
a.__next__()   # 32
a.__next__()   # Raise StopIteration Exception
```



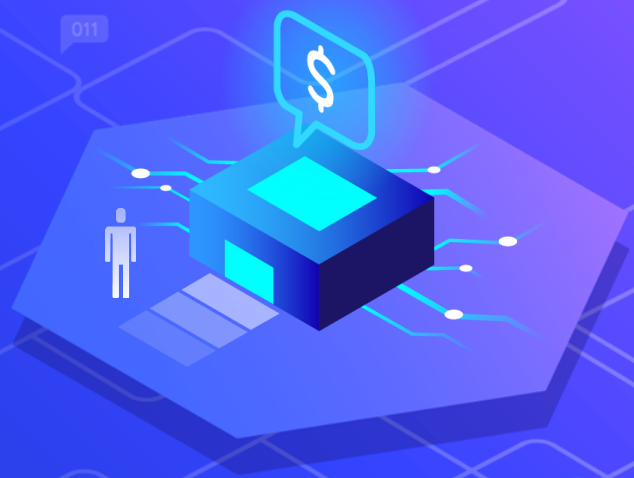
Generator

Much Better

no need to implement `__iter()__` and `__next__()`

```python
def my_gen():
    n = 1
    print('This is printed first')
    yield n

    n += 1
    print('This is printed second')
    yield n

    n += 1
    print('This is printed at last')
    yield n

a = my_gen()
a.__next__() # This is printed first 1
a.__next__() # This is printed second 2
a.__next__() # This is printed at last 3
a.__next__() # Raise StopIteration Exception

##########################################

for item in my_gen():
    print(item)

"""
This is printed first
1
This is printed second
2
This is printed at last
3
"""
```

Questions ?!



Thanks!

>_ Live long and prosper