הטכניון
היחידה ללימודי המשך
ולימודי חוץ עייש עזריאלי

Object Oriented Programming (OOP)

TECHNION
Azrieli Continuing Education and
External Studies Division

Agenda

1. Inheritance
2. Polymorphism
3. Duck typing

Full class:

```python
import math

class Circle:
  "A circle with a center point and a radius"

  def __init__(self, center, radius):
    self.center = center
    self.radius = radius

  def area(self):
    return math.pi*(self.radius)**2

  def circumference(self):
    return 2* math.pi * self.radius

  def calculate_distance(self, circle):
    center_distance = math.sqrt(sum(
        (px - qx) ** 2.0 for px, qx in zip(self.center, circle.center)))
    return center_distance - self.radius - circle.radius

circle1 = Circle(center=(10, 15), radius=3)
circle2 = Circle((0, 0), 6)
distance = circle1.calculate_distance(circle2)
print(f'Circles distance: {distance}')
```

Document your class!

Constructor and attributes

Methods

```
Circles distance: 9.027756377319946
```

Review:

Naming convention

- Variables & Functions
  - All small letters.
  - Separate words with _.
  - Don’t start with a number.

- **Classes:**
  - **Class name with CamelCase.**
  - **Attributes – like regular variables.**
  - **Methods – like regular functions.**
  - **One class in one file.**
  - **File name as the class name.**

Inheritance

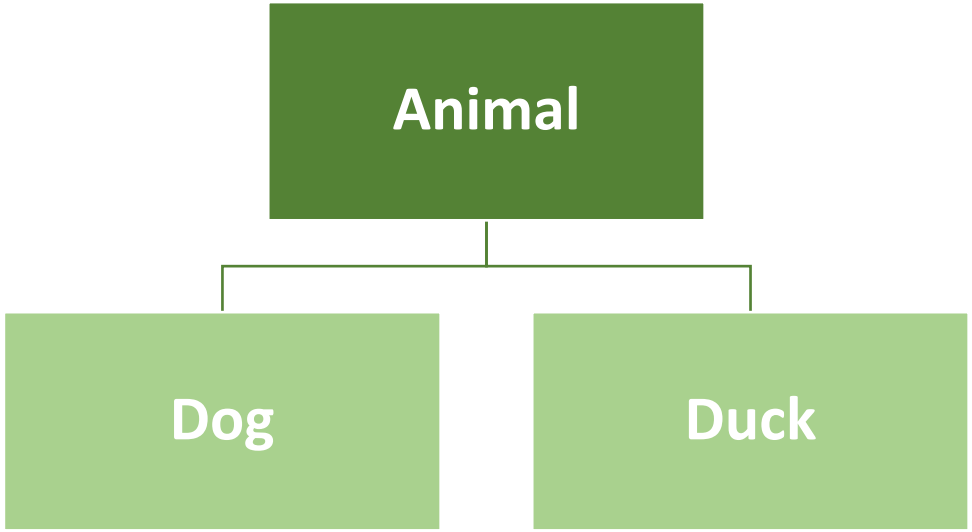
Animals example

- Common features of dog and a duck:
  - Color.
  - Legs.
  - Both can walk.

- Differences:
  - Dog can bark,
  - Duck can quack.

IS-A relationship

- Dog is-an animal.
- Duck is-an animal.



Inheritance

- Put common features in the parent class (animal).
- Specific features and implementations in the child class (dog/duck).

```
class Animal:
  def __init__(self,legs, color):
    self.legs = legs
    self.color = color

  def walk(self):
    print('Walking...')

class Dog(Animal):
  def growl(self):
    print("I'm a good boy until I get mad. Grrrr")

class Duck(Animal):
  def quack(self):
    print("quack quack quack")
```

```python
class Animal:
  def __init__(self,legs, color):
    self.legs = legs
    self.color = color

  def walk(self):
    print('Walking...')

class Dog(Animal):
  def growl(self):
    print("I'm a good boy until I get mad. Grrrr")

class Duck(Animal):
  def quack(self):
    print("quack quack quack")
```

```python
daffy = Duck(2, 'black')
print('Duck')
print(f'legs: {daffy.legs}')
print(f'color: {daffy.color}')
daffy.walk()
daffy.quack()
```

```
Duck
legs: 2
color: black
Walking...
quack quack quack
```

Inheritance

Override

- If a child class have a method with the same name as the parent class- the parent method will be overwritten.
- Meaning- the parent method won't be used at all!

```
[11]  class Animal:
        def __init__(self,legs, color):
          self.legs = legs
          self.color = color

        def walk(self):
          print('Walking...')

      class Dog(Animal):
        def __init__(self,legs, color, tail):
          self.tail = tail

        def growl(self):
          print("I'm a good boy until I get mad. Grrrr")
```

```
hilik = Dog(4, 'brown', True)
print('Dog')
print(f'legs: {hilik.legs}')
print(f'tail: {hilik.tail}')
```

```
Dog
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-12-918888117357> in <module>()
      1 hilik = Dog(4, 'brown', True)
      2 print('Dog')
----> 3 print(f'legs: {hilik.legs}')
      4 print(f'tail: {hilik.tail}')

AttributeError: 'Dog' object has no attribute 'legs'
```

Super()

- If a child class wants to refer to his parent's attributes/methods, we can use super():

```
super().method(value, value, …)
super().attribute
```

```
class Animal:
  def __init__(self,legs, color):
    self.legs = legs
    self.color = color

  def walk(self):
    print('Walking...')

class Dog(Animal):
  def __init__(self,legs, color, tail):
    super().__init__(legs, color)
    self.tail = tail

  def growl(self):
    print("I'm a good boy until I get mad. Grrrr")
```

```
hilik = Dog(4, 'brown', True)
print('Dog')
print(f'legs: {hilik.legs}')
print(f'tail: {hilik.tail}')
```

```
Dog
legs: 4
tail: True
```

Polymorphism- allows different objects to be treated the same way, as they implement the required methods

Duck typing - is a concept in Python where an object's suitability is determined by the presence of certain methods or properties, rather than its type. As the saying goes: *"If it walks like a duck and quacks like a duck, it's a duck."* Python focuses on the behavior of an object rather than its class.

```python
class Dog:
    def sound(self):
        return "Bark"

class Duck:
    def sound(self):
        return "Quack"

def make_sound(animal):
    return(animal.sound())


dog = Dog()
duck = Duck()
```

```python
animals = [dog,duck]
for animal in animals:
    print(f" animal is {make_sound(animal)}ing")
```

```
animal is Barking
animal is Quacking
```

Summary

OOP - Programing paradigm.

Class

Define your own type, with attributes and methods.

Self, init constructure.

Object

An instance of a class.

Access inner attributes and methods with “.” operator.