JÖNKÖPING UNIVERSITY

School of Engineering

OOP IN PYTHON

Peter Larsson-Green
Jönköping University
Autumn 2018

MODELLING

The data

```
ages = [43, 47, 10, 7, 3]
```

Computations

```
def average(numbers):
    return sum(numbers)/len(numbers)
```

User Interface

```
print("Average age: "+str(average(ages)))
```

Object-Oriented Programming keeps data and functions together.

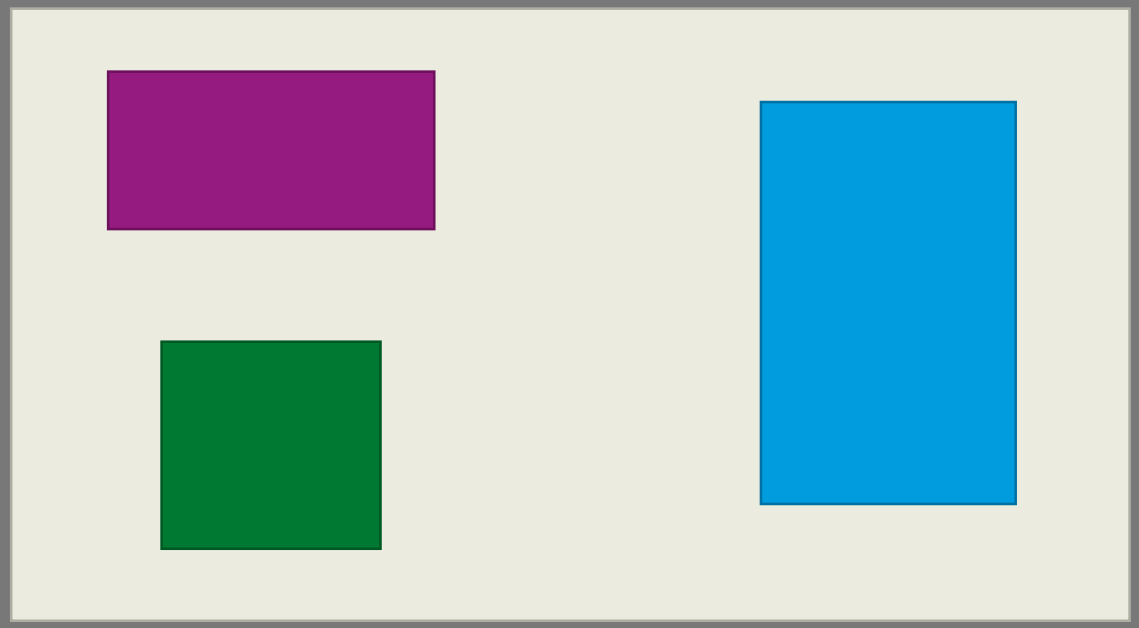
MODELLING

```
humans = [
  {"name": "Alice", "age": 10},
  {"name": "Bob", "age": 15}
]
def get_average_age(humans):
  sum = 0
  for human in humans:
    sum += human["age"]
  return sum / len(humans)
```

```
pets = [
  {"type": "dog", "age": 2},
  {"type": "cat", "age": 4}
]
def get_average_age(pets):
  sum = 0
  for pet in pets:
    if pet["type"] == "dog":
      sum += pet["age"] * 7
    else:
      sum += pet["age"] * 4
  return sum / len(pets)
```

CLASSES AND OBJECTS

A class is a description of/template for *something*.



Screen.

Use a class to describe a rectangle:

- Width & height
- Position
- Color

Create 3 instances to represent your actual rectangles.

Objects are instances of classes.

YOU HAVE ALREADY USED OOP

All values in Python are objects.

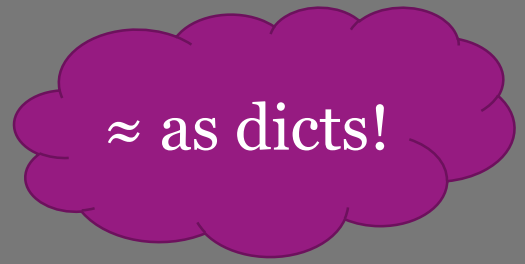
```
>>> a = 123
>>> type(a)
<class 'int'>
>>> b = "abc"
>>> type(b)
<class 'str'>
>>> c = [1, 2]
>>> type(c)
<class 'list'>
```

CLASSES AND OBJECTS

Objects are instances of classes.

Consists of:

- Data fields (store values).



A class is a description of/template for objects.

Consists of:

- Methods (defines what you can do with the objects).
  - Constructor.
  - Operations.

A SIMPLE CLASS

```
class MyClass:
  pass
```

The name of the class.

```
object_a = MyClass()
object_a.three = 3
print(object_a.three)
object_b = MyClass()
object_b.three = "three"
print(object_b.three)
print(object_a.three)
```

Creates a new instance of the class.

Prints: 3

Prints: three

Prints: 3

THE CONSTRUCTOR

```
class Circle:
  pass
```

```
class Circle:
  def __init__(self):
    self.radius = 10
```

```
circle_a = Circle()
circle_a.radius = 10
print(circle_a.radius)
circle_b = Circle()
circle_b.radius = 10
print(circle_b.radius)
```

```
circle_a = Circle()
print(circle_a.radius)
circle_b = Circle()
print(circle_b.radius)
```

THE CONSTRUCTOR

```
class Circle:
  def __init__(self, radius):
    self.radius = radius
```

```
class Circle:
  def __init__(self):
    self.radius = 10
```

```
small_circle = Circle(10)
print(small_circle.radius)
big_circle = Circle(200)
print(big_circle.radius)
```

```
circle_a = Circle()
print(circle_a.radius)
circle_b = Circle()
print(circle_b.radius)
```

METHODS

```
class Circle:
  def __init__(self, radius):
    self.radius = radius
  def get_area(self):
    return (self.radius ** 2) * 3.14
  def get_perimeter(self):
    return self.radius * 2 * 3.14
```

```
my_circle = Circle(30)
print(my_circle.get_area())      # Prints 2826.
print(my_circle.get_perimeter()) # Prints 188.4.
```

EXAMPLE

```
class Rectangle:
  def __init__(self, width, height):
    self.width = width
    self.height = height
  def get_area(self):
    return self.width * self.height
  def get_perimeter(self):
    return self.width*2 + self.height*2
```

```
rectangle = Rectangle(10, 20)
print(rectangle.get_area())      # Prints 200.
print(rectangle.get_perimeter()) # Prints 60.
```

EXAMPLE - CALCULATOR LAB 3

```
Enter initial memory value: 0
Enter operation (add/sub/quit): add
Enter operand: 20
20 is stored in memory.
Enter operation (add/sub/quit): sub
Enter operand: 5
15 is stored in memory.
Enter operation (add/sub/quit): quit
```

```
calc = Calculator(0)
calc.add(20)
calc.subtract(5)
calc.get_memory_value()
```

We can use a class to represent the calculator.

- Need to keep track of the memory value.
- Need to be able to change the memory value (add/sub).

EXAMPLE - CALCULATOR LAB 3

```
class Calculator:
  def __init__(self, initial_memory_value):
    self.memory_value = initial_memory_value
  def add(self, operand):
    self.memory_value += operand
  def subtract(self, operand):
    self.memory_value -= operand
  def get_memory_value(self):
    return self.memory_value
```

```
calc = Calculator(0)
calc.add(20)
calc.subtract(5)
calc.get_memory_value()
```

EXAMPLE - CALCULATOR LAB 3

```
initial_memory_value = int(input("Enter initial memory value: "))
calculator = Calculator(initial_memory_value)
operation = ""
while operation != "quit":
  operation = input("Enter operation (add/sub/quit): ")
  if operation != "quit":
    operand = int(input("Enter operand: "))
    if operation == "add":
      calculator.add(operand)
    elif operation == "sub":
      calculator.subtract(operand)
    print(str(calculator.get_memory_value())+" is stored in memory.")
```

DICE EXAMPLE

```
from random import randint
dice_value = randint(1, 6)
print("You got: "+str(dice_value)+".")
dice_value = randint(1, 6)
print("You got "+str(dice_value)+".")
```

```
class Dice:
  def __init__(self):
    self.roll()
  def roll(self):
    from random import randint
    self.number_of_pips = randint(1, 6)
```

```
dice = Dice()
print("You got: "+str(dice.number_of_pips)+".")
dice.roll()
print("You got: "+str(dice.number_of_pips)+".")
```

DICE EXAMPLE

```
class SetOfDice:
  def __init__(self, number_of_dice):
    self.dice_list = []
    for i in range(number_of_dice):
      self.dice_list.append(Dice())
  def get_number_of_ones(self):
    count = 0
    for dice in self.dice_list:
      if dice.number_of_pips == 1:
        count += 1
    return count
```

```
set_of_dice = SetOfDice(5)
if set_of_dice.get_number_of_ones() == 0:
  print("Nice!")
else:
  print("That's too bad.")
```

ROOM EXAMPLE

```
class Room:
  def __init__(self, name, side_length_1, side_length_2):
    self.name = name
    self.side_length_1 = side_length_1
    self.side_length_2 = side_length_2
 def get_area(self):
    return self.side_length_1 * self.side_length_2
```

```
room1 = Room("Kitchen", 7, 5)
print(room1.name+" is "+str(room1.get_area()+" m^2 big."))
room2 = Room("Bed Room", 3, 4)
print(room2.name+" is "+str(room2.get_area()+" m^2 big."))
```

HOUSE EXAMPLE

```
class House:
  def __init__(self):
    self.rooms = []
  def add_room(self, room):
    self.rooms.append(room)
  def get_area(self):
    sum = 0
    for room in self.rooms:
      sum += room.get_area()
    return sum
```

```
my_house = House()
my_house.add_room(Room("Kitchen", 7, 5))
my_house.add_room(Room("Bed Room", 3, 4))
area = my_house.get_area()
print("Area of my house: "+str(area)+".")
```