

# Classes and Objects



**SoftUni Team**  
Technical Trainers



**SoftUni**



**Software University**

<https://softuni.bg>

[sli.do](https://sli.do)

**#fund-python**

1. Object Oriented Programming (OOP)
2. Classes
3. Objects
4. Class Attributes and Instance Methods





# Object Oriented Programming

OOP

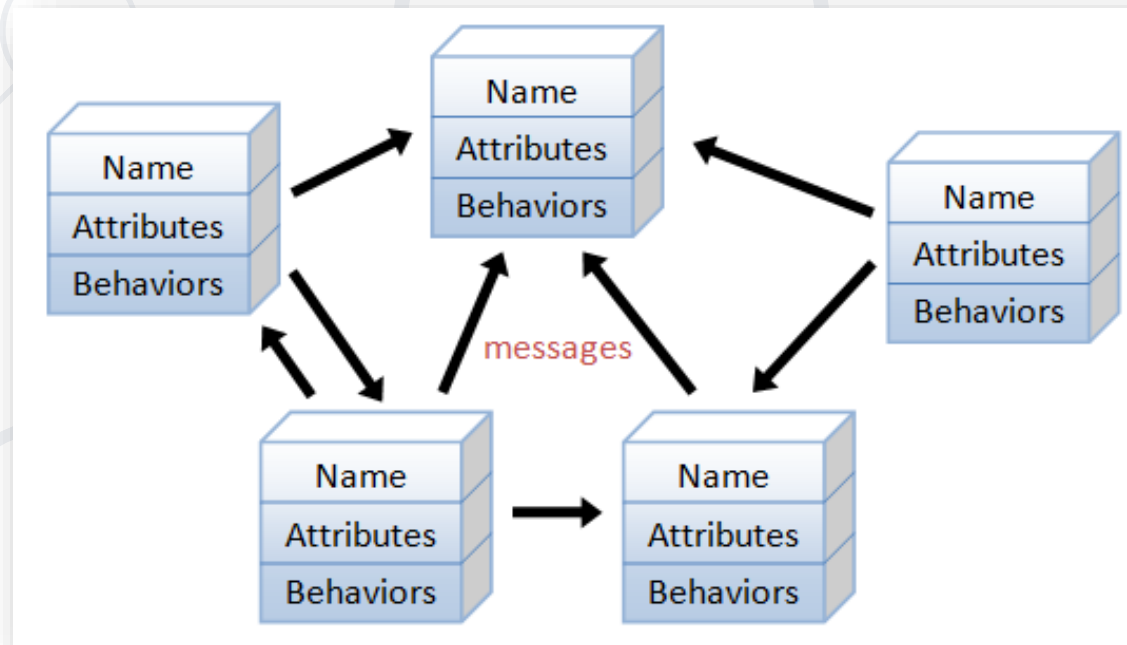
# What is OOP?

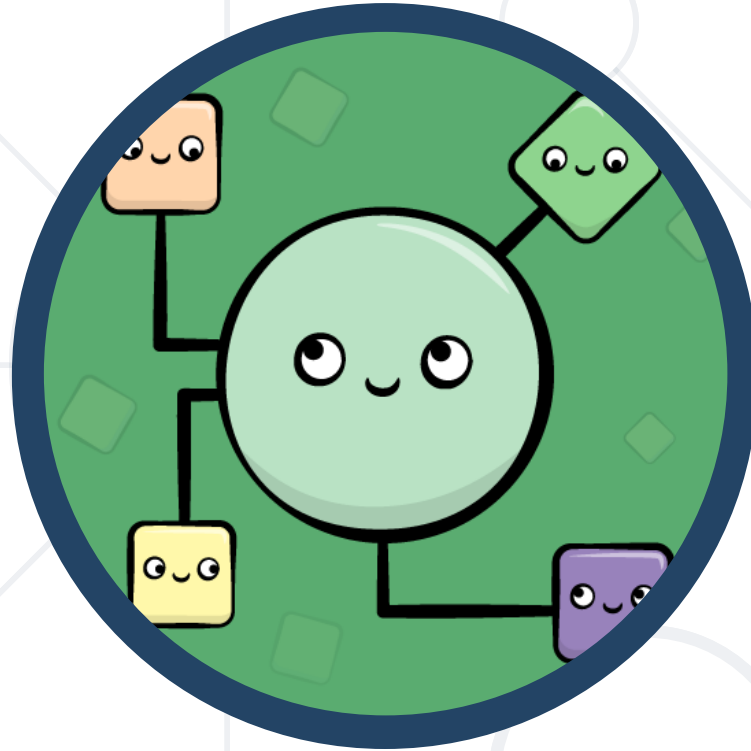
- OOP is a programming **paradigm**
- Provides a means of **structuring** programs so that properties and behaviors are bundled into individual **objects**
- Objects are at the center of the OOP paradigm



# Example

- An object-oriented program consists of objects that **interact** with each other





# Classes

Template for Objects

# Class Definition

- A **class** is like a **blueprint** for creating **objects**
- Classes provide a means of bundling data and functionality together
- Each class instance can have **attributes** attached to it
- Class instances can also have **methods** for modifying its state





# The `__init__()` Function

- All classes create objects, and all objects contain characteristics called **attributes**
- The `__init__()` method initializes an object's initial attributes by giving them their default value
- The double leading and trailing underscore is used for **special variables** or **methods**



# The self Variable

- The **self** parameter is a reference to the current **instance** of the class
- Used to access variables that belong to the class
- When defining an **instance** method, the first parameter of the method should always be **self**



# Class Example

- Create a **class Person** that has, **first name**, **last name**, and **age**

```
class Person:
    def __init__(self, first_name, last_name, age=0):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
```

Default value



- When a parameter is **optional**, but a **default value** that it should have

- Create a class with the name **"Comment"**
- The **\_\_init\_\_** method should accept **3 parameters**
  - username
  - content
  - likes (optional, 0 by default)
- Use the exact names of your variables

```
class Comment:
    def __init__(self, username, content, likes=0):
        self.username = username
        self.content = content
        self.likes = likes
```



# Objects

Instances of Classes

# Object Definition

- **Instance** is a concrete object of the type of the class
- A class is like a form or a template
  - It defines the needed information
- You can fill out multiple copies, but without the form, you will not know the required information



# Object Example

- Using the **Person** class we created, here we have an example of the instance of that class

```
me = Person("Peter", "Johnson", 25)
print(me.first_name) # Peter
print(me.last_name)  # Johnson
print(me.age)        # 25
```



Accessing object  
**attributes**



- You can change the values of the attributes of an object after initialization

```
me = Person("Peter", "Johnson", 25)
me.age += 1
print(me.age) # 26
```

- The change will be made only for that instance of the class

# Problem: Party

- Create a **class Party** that only has an attribute called **people**
- The **\_\_init\_\_** method should not accept any parameters
- You will be given names of people (on separate lines) until you receive the command "**End**"
- Print 2 lines:
  - **"Going: "** + all the people separated by comma and space
  - **"Total: {total\_people\_going}"**

```
class Party:
    def __init__(self):
        self.people = []

party = Party()
line = input()
while line != "End":
    party.people.append(line)
    line = input()
print(f"Going: {' '.join(party.people)}")
print(f"Total: {len(party.people)}")
```



# **Class Attributes and Instance Methods**

# Class Attributes

- While instance attributes are specific to each object, class attributes are the same for all instances

```
class Person:
    species = "mammal"
    def __init__(self, name, age):
        self.name = name
        self.age = age
me = Person("Peter", 25) # me.species = "mammal"
you = Person("John", 44) # you.species = "mammal"
```



# Instance Methods

- Instance methods are defined **inside a class** and are used to get the **contents of an instance**
- They can also be used to perform **operations** with the **attributes** of our objects
- Like the **`__init__`** method, the first argument should always be **`self`**



```
class Person:
    def __init__(self, first_name, last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
    def get_full_name(self):
        return f"{self.first_name} {self.last_name}"
me = Person("Peter", "Johnson", 64)
print(me.get_full_name()) # Peter Johnson
```

- Create a class **Email** as described in the lab description
- You will receive some emails until you receive "**Stop**" (separated by a single space)
- The **first** will be the **sender**, the **second** one – the **receiver** and the **third** one – the **content**
- On the final line you will be given the indices of the **sent emails** separated by comma and space. For each email print:  
`"{sender} says to {receiver}: {content}.`  
`Sent: {is_sent}"`



# Solution: Email (the Class)

```
class Email:
    def __init__(self, sender, receiver, content):
        self.sender = sender
        self.receiver = receiver
        self.content = content
        self.is_sent = False
    def send(self):
        self.is_sent = True
    def get_info(self):
        return f"{self.sender} says to {self.receiver}:  
{self.content}. Sent:  
{self.is_sent}"
```

# Solution: Email (the Logic)

```
emails = []
line = input()
while line != "Stop":
    tokens = line.split(" ")
    email = Email(tokens[0], tokens[1], tokens[2])
    emails.append(email)
    line = input()
    send_emails = [int(x) for x in input().split(", ")]
for x in send_emails:
    emails[x].send()
# TODO: print the emails
```

- Create a **class Zoo** as described in the lab description
- On the first line you will receive the **name** of the zoo
- On the second line you will receive the number **n**
- On the next **n lines** you will receive animal info in the format: "**{species} {name}**". Add the animal to the zoo
- On the final line you will receive a **species**
- Print all the info for that species and the total count of animals

# Solution: Zoo (the Class)

```
class Zoo:
    def __init__(self, name):
        self.name = name
        # TODO: create 3 Lists (mammals, fishes, birds)
    def add_animal(self, species, name):
        # TODO: add the name to the given species
    def get_info(self, species):
        # TODO: create the resulting string and return it
```

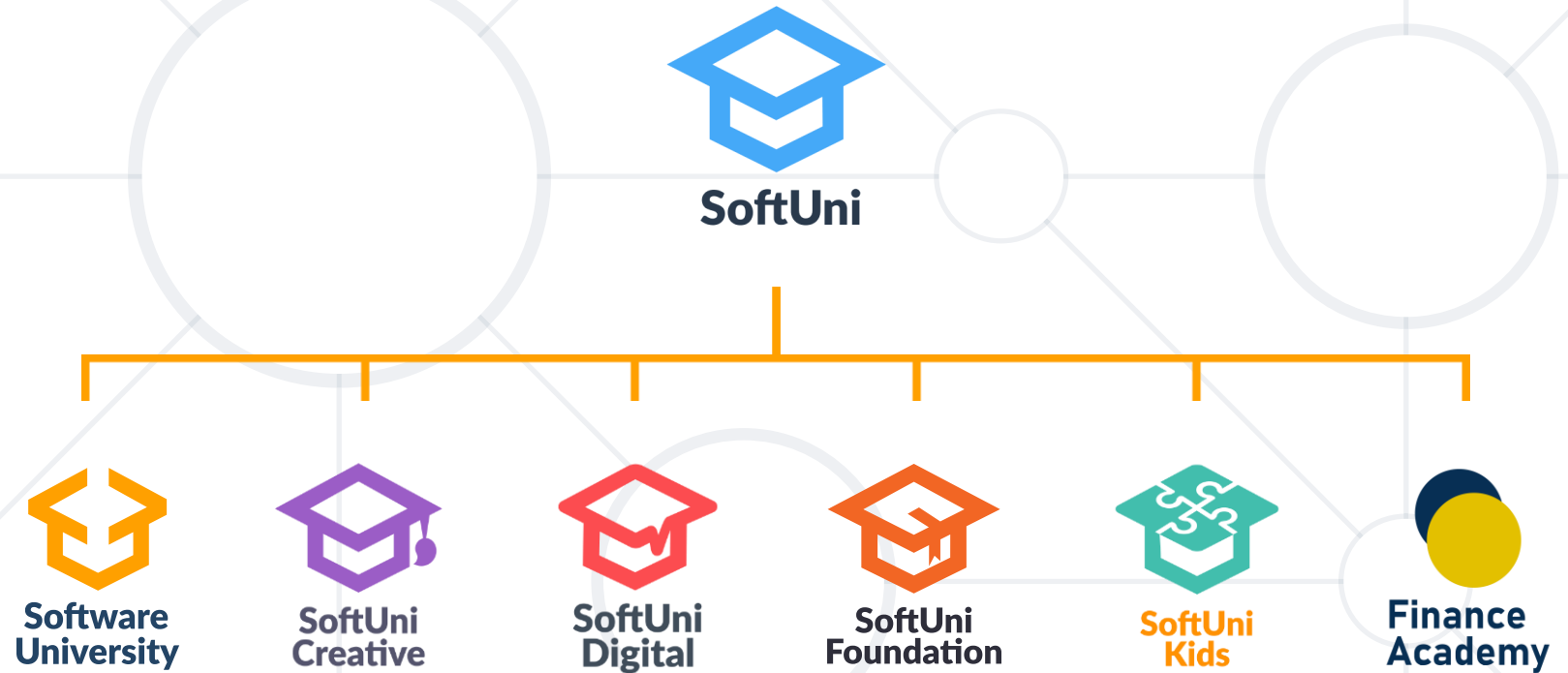
# Solution: Zoo (the Logic)

```
zoo_name = input()
zoo = Zoo(zoo_name)
count = int(input())
total_animals = 0
for i in range(count):
    # Read the input
    # Add the new animal
    # Update the total_animals variable
    info = input()
    print(zoo.get_info(info))
```

- What does **OOP** mean
- What **classes** are and how to create them
- What **instances** are and how to create them
- What **attributes** and **instance methods** are



# Questions?



# SoftUni Diamond Partners





- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [softuni.org](http://softuni.org)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](http://forum.softuni.bg)



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

