

# First Steps in Object-Oriented Programming



SoftUni Team  
Technical Trainers



**SoftUni**



Software University

<https://softuni.bg>

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

**#python-advanced**

1. Project Architecture
2. Scope and Namespace
3. Basics of OOP
  - Object
  - Class
  - Instance
4. Creating and Using Classes





# **Project Architecture**

## **Splitting Code into Logical Parts**

# Splitting Code into Functions

- We use **methods** to split code into functional blocks
  - Improves code **readability**
  - Allows for easier **debugging**

```
for move in moves:  
    for row in range(len(room)):  
        for col in range(len(room[row])):  
            if room[row][col] == 'b':  
                ...
```



```
for move in moves:  
    def move_enemies()  
    def killer_check()  
    def move_player(move)
```

# Splitting Code into Functions

- A **single** function should complete a **single task**

```
do_magic ( ... )  
deposit_or_withdraw ( ... )  
deposit_and_get_balance ( ... )  
parse_data_and_return_result ( ... )
```



```
withdraw ( ... )  
deposit ( ... )  
get_balance ( ... )  
to_string ( ... )
```



# Problem: Rhombus of Stars

- Draw on the console a rhombus of stars with a size of **n**

$n = 3$



```
*  
* *  
* * *  
* *  
*  

```

$n = 2$



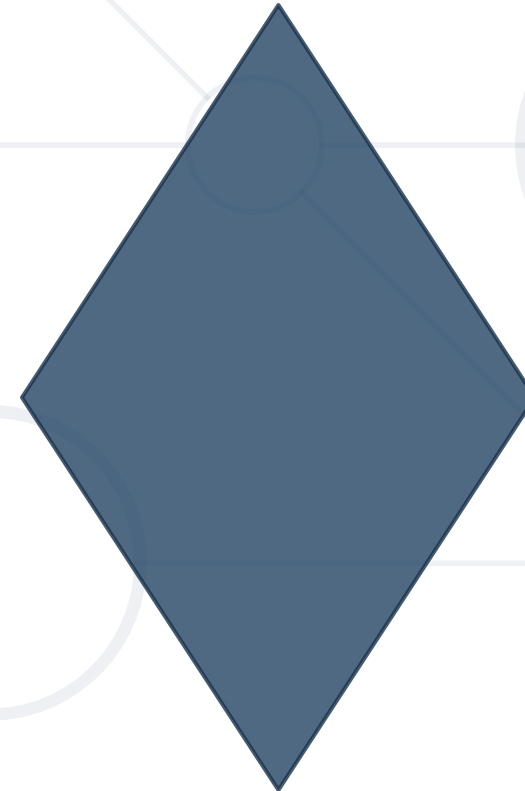
```
*  
* *  
*  

```

$n = 1$



```
*
```



# Solution: Rhombus of Stars

```
def print_row(size, star_count):  
    for row in range(size - star_count):  
        print(" ", end="")  
    for row in range(1, star_count):  
        print("*", end=" ")  
    print("*")
```

Reusing code

```
size = int(input())  
for star_count in range(1, size):  
    print_row(size, star_count)  
for star_count in range(size, 0, -1):  
    print_row(size, star_count)
```





# **Scope and Namespace**

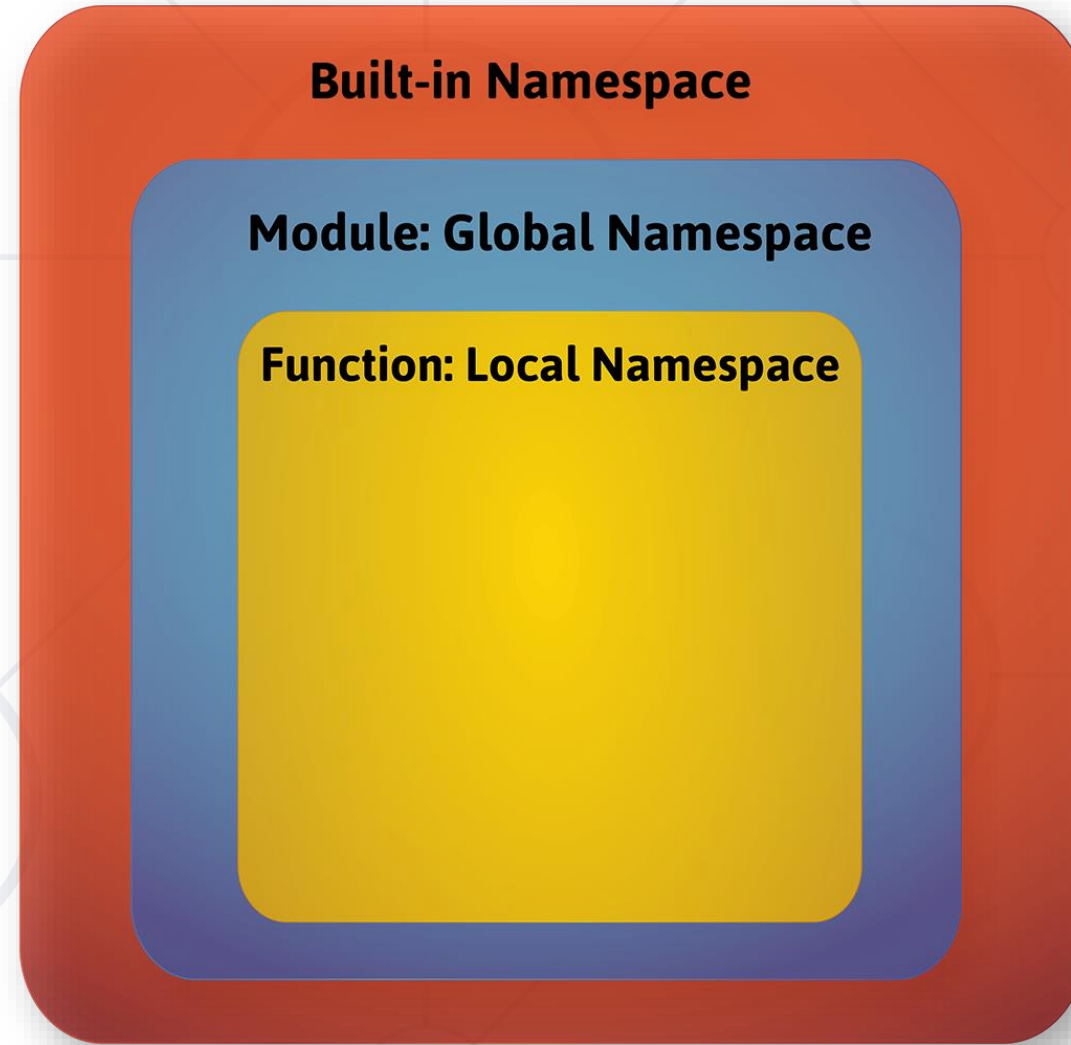
Local, Global and Built-In Namespace

# What is Namespace?

- A mapping from names to objects
- Examples:
  - **Built-in** names, for example, the **abs()** function
  - **Global** names in a module
  - **Local** names on a function invocation
- There is no relation between names in different **namespaces**



# Namespaces Order



# What is a Scope?

- A region in a program where a **namespace** is directly accessible
- In most of cases there are at least three nested **scopes**:
  - The **innermost** is checked first
  - The scopes of any **enclosing functions**
  - The next-to-last scope (module's **global** names)
  - The outermost (**built-in** names)



# Scopes Example

```
def scopes():  
    def local_scope():  
        text = "local text"
```

Local Scope

```
def nonlocal_scope():  
    nonlocal text  
    text = "nonlocal text"
```

Nonlocal Scope

```
def global_scope():  
    global text  
    text = "global text"
```

Global Scope

# Problem: Scope Mess

- Fix the provided code, so it prints the expected result
- Download the code from [here](#)

```
# current output  
global  
outer: local  
inner: nonlocal  
outer: local  
global
```

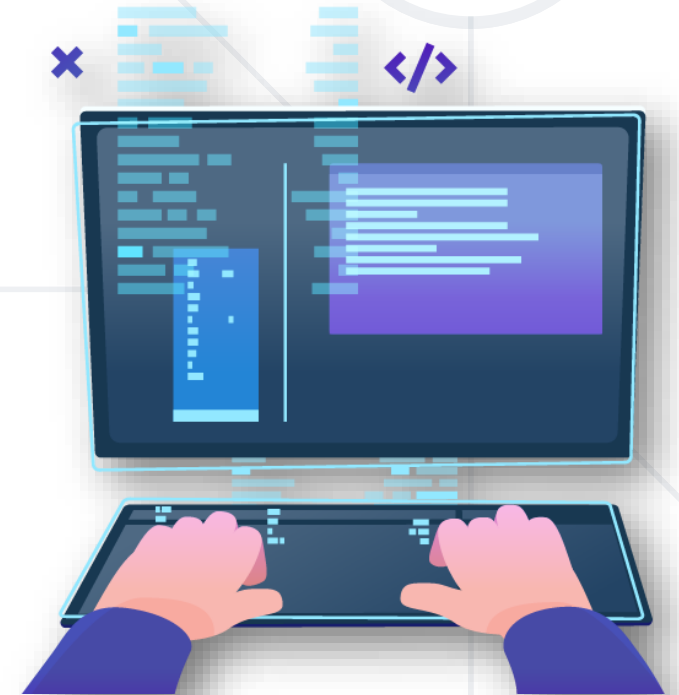
```
# expected output  
global  
outer: local  
inner: nonlocal  
outer: nonlocal  
global: changed!
```

# Solution: Scope Mess

- Here are the changes that need to be made

```
def inner():  
    nonlocal x  
    x = "nonlocal"  
    print("inner:", x)
```

```
def change_global():  
    global x  
    x = "global: changed!"
```





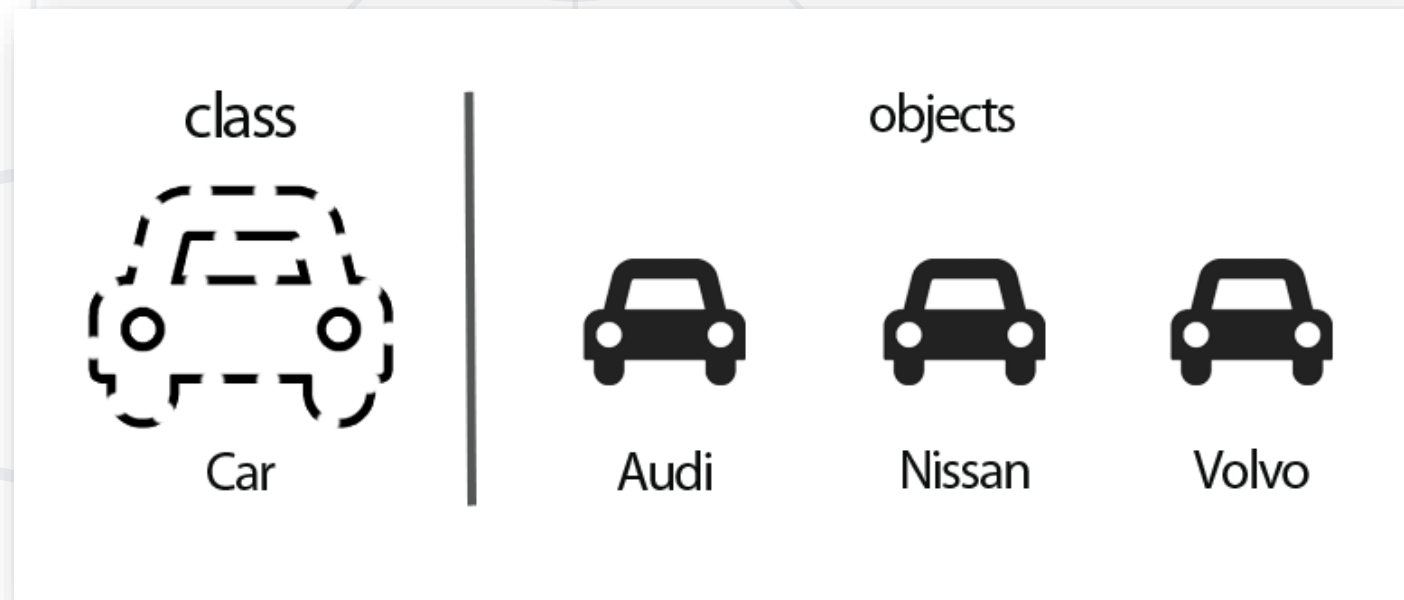
# Basics of OOP

Building Data Functionality Together



# What is an Object-Oriented Programming?

- It is the **most popular** programming paradigm
- It relies on the concept of **classes and objects**
- A **class** is used to create an **individual instance** of an **object**



# Advantages of OOP

- Provides a **clear program structure** and a **clean code**
- **Reduces** complexity
- Make it **easy to write** a reusable code
- Could **test** each behavior of an object **separately**
- Facilitates **easy maintenance** and **modification** of existing code



# Objects in Python

- **Everything** in Python is **an object** and has **a type**
  - 10.5
  - "Python"
  - [1, 2, 3, 4]
  - {"name": "Peter", "age": 26}
- We could **create** as many objects as we like, **manipulate** them, or **remove** them



- **Create** an object of type list

```
numbers = [1, 2, 3, 4, 8, 10]
```

The internal representation of the list is private

- **Manipulate** the object by adding an element

```
numbers.append(5)  
print(numbers) # [1, 2, 3, 4, 8, 10, 5]
```

You need to know how to use its methods

- **Remove** the object

```
del numbers  
print(numbers) # Error
```

# What is an Object?

- Object is a **data abstraction** that captures an **internal representation** and **an interface**
- The internal representation should be **private**
- The interface **defines behaviors** but **hides implementation**



# Characteristics of an Object

- **State**

- Help to distinguish an object from other objects
- A phone could have a color, a size, a weight

- **Behavior**

- The tasks that an object performs
- A phone could turn on, turn off



# What is a Class?

- The class is a **blueprint that defines the nature** of a future object
- In Python, a class is created by the keyword **class**



Class Name

```
class Phone:
    def __init__(self, color, size):
        self.color = color
        self.size = size


    def turn_on(self):
        return 'The phone is turned on'
```

State

Behavior

# What is an Instance?

- **Specific realization** of an object of a certain class
- The creation of an instance is called **instantiation**



```
class Phone:
    def __init__(self, color, size):
        self.color = color
        self.size = size

phone = Phone("blue", 4.7)
```

Instance





# **Creating and Using Classes**

- The keyword **class** defines a **new type**

```
class Person:  
    pass
```

Type Person

- We define **the state** of the object using **attributes**

```
class Person:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

Special Method

Attributes

# Problem: Class Book

- Create a class called **Book**
- Upon initialization, it should receive a **name**, **author**, and **pages** (number)
- Submit only the class in the judge system
- Use test code to test your code

```
book = Book("My Book", "Me", 200)
print(book.name)
print(book.author)
print(book.pages)
```



```
My Book
Me
200
```

```
class Book:  
    def __init__(self, name, author, pages):  
        self.name = name  
        self.author = author  
        self.pages = pages
```



# Method

- We define **the behavior** of the object using **methods**
- It is like a function, that **works only within a class**



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

    def sleep(self):
        return "sleeping.."

animal = Animal("cat")
print(animal.sleep()) # sleeping..
```

Defining a Method

# Using a Class

- Using a class means **creating new instances** of objects and **executing operations** on the instances

```
class Person():  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def eat(self):  
        return 'eating..'
```

```
person = Person()  
print(person.eat()) # eating..
```

Instance



# Problem: Car

- Create a class **Car** that receives **name**, **model**, and **engine** upon initialization
- It should have **a method** called **get\_info()** which returns **'This is {name} {model} with engine {engine}'**
- Submit only the class in the judge system

```
car = Car("Kia", "Rio", "1.3L B3 I4")  
print(car.get_info())
```



```
This is Kia Rio with engine 1.3L B3 I4
```

```
class Car:
    def __init__(self, name, model, engine):
        self.name = name
        self.model = model
        self.engine = engine

    def get_info(self):
        return f'This is {self.name} {self.model} ' \
               f'with engine {self.engine}'
```



# Problem: Music

- Create a class **Music** that receives **title**, **artist**, and **lyrics** upon initialization
- It should have **2 methods**
  - **print\_info()** - returns 'This is {title} from {artist}'
  - **play()** - returns the lyrics
- Submit only the class in the judge system
- Test your code with your own examples

```
class Music:
    def __init__(self, title, artist, lyrics):
        self.title = title
        self.artist = artist
        self.lyrics = lyrics

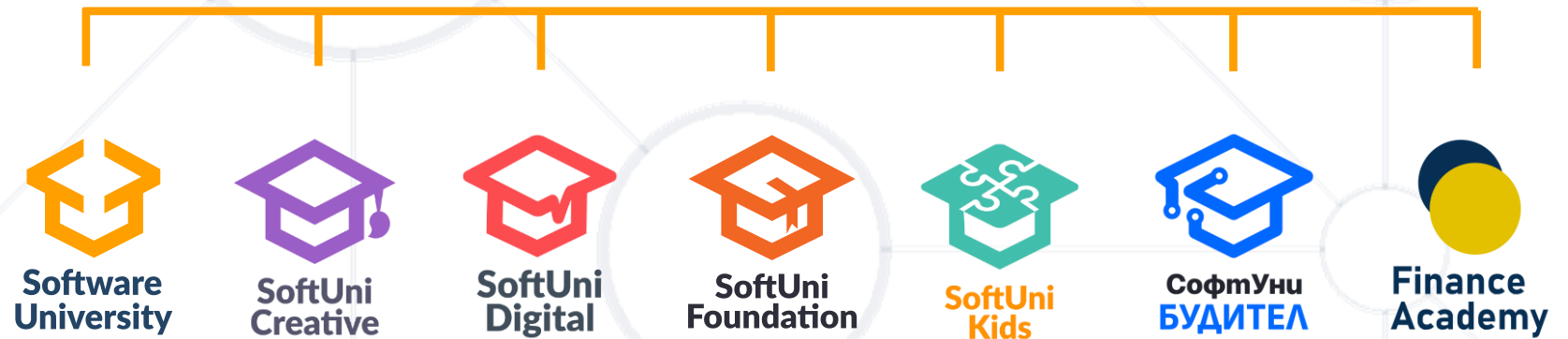
    def print_info(self):
        return f'This is "{self.title}" from "{self.artist}"'

    def play(self):
        return self.lyrics
```

- **OOP** relies on the concept of classes and objects
- **Object** is a data abstraction that captures an internal representation and an interface
- **Class** is a blueprint that defines the nature of a future object
- **Instance** is a specific realization of any object



# 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)



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

