Functions & Modular Code

Python Course · Module 5

ElnurBDa

Functions — Introduction

Functions group reusable code into named blocks.

Benefits:

- Avoid repetition
- Improve readability
- Easier debugging
- Modular structure

Basic structure:

```python
def name():
    pass
```

Creating Functions

```python
def greet():
    print("Hello!")
```

Call:

```python
greet()
```

Functions must be defined **before** calling.

_____

Parameters

```python
def greet(name):
    print("Hello,", name)
```

Multiple parameters:

```python
def add(a, b):
    return a + b
```

_____

Return Values

```python
def square(x):
    return x * x
```

Functions without return return None.

Multiple Returns

```python
def values():
    return 10, 20, 30
```

Unpacking:

```python
a, b, c = values()
```

Variable Scope

```python
x = 10      # global

def func():
    y = 5   # local
```

Local variables override global ones inside functions.

_____

global Keyword

```python
count = 0

def inc():
    global count
    count += 1
```

Use sparingly; global state makes code harder to manage.

Lambda Functions

```python
square = lambda x: x * x
```

Often used inline:

```python
add = lambda a, b: a + b
```

Higher-Order Functions

```
nums = [1, 2, 3, 4]
squares = list(map(lambda x: x*x, nums))
```

Using normal function:

```
def sq(x): return x*x
```

_____

Modules — Importing

```python
import math
math.sqrt(16)
```

Selective:

```python
from math import sqrt
```

Rename:

```python
import math as m
```

Built-in Modules

Examples:

- math
- random
- os
- sys
- datetime

Example:

```python
import random
x = random.randint(1, 10)
```

Creating Your Own Module

File:

```
mymath.py
```

Content:

```python
def add(a, b):
    return a + b
```

Usage:

```python
import mymath
mymath.add(2, 3)
```

Organizing Code

```
project/
├── main.py
├── utils.py
└── math_ops.py
```

```python
from utils import greet
from math_ops import add
```

Example

```
 1 # utils.py
 2 def greet(name):
 3     return f"Hello {name}"
 4
 5 # math_ops.py
 6 def multiply(a, b):
 7     return a * b
 8
 9 # main.py
10 from utils import greet
11 from math_ops import multiply
12
13 print(greet("Elnur"))
14 print(multiply(4, 5))
```

Mini Task

Create a project with:

1. main.py
2. converter.py with:
   ◦ km_to_miles(km)
   ◦ celsius_to_fahrenheit(c)
3. Import into main.py
4. Ask user input and print converted values

_____

Objects & Classes — Why They Matter

Functions are perfect for **single actions**. Classes help when you need to model **entities** (users, sensors, tickets) that carry both data and behavior.

- **Object** = "thing" with attributes + methods
- **Class** = blueprint describing what every object of that type should have

Analogy:

- Class = recipe for User
- Object = actual user baked from that recipe

Use classes when you keep passing the same fields around or when multiple behaviors belong to the same piece of data.

_____

Defining a Class

```python
class User:
    def __init__(self, username, email):
        self.username = username
        self.email = email
        self.active = True

    def deactivate(self):
        self.active = False

user = User("alice", "alice@example.com")
print(user.username, user.active)
user.deactivate()
print(user.active)
```

———————————————— [finished] ————————————————

```
alice True
False
```

Notes:

- __init__ runs every time you call User(...)
- self points to the current object so you can store/read attributes
- Methods are normal functions defined inside the class

Attributes vs Methods

Concept	Purpose	Example
Attribute	Data stored on the object	user.email, user.active
Method	Action using that data	user.deactivate()
Class attr	Shared for all instances	class User: role = "student"

Guidelines:

- Keep attributes small and descriptive
- Use methods when behavior depends on the object's state
- Return values from methods when you need reusable data (describe())

_____

Classes vs Simple Functions

```
Prefer simple functions when:
  - Logic is stateless
  - You just manipulate inputs/outputs quickly
  - The helper stands on its own (e.g., convert units)

Prefer classes when:
  - You repeatedly use the same bundle of data
  - Multiple behaviors belong to that bundle
  - You want to hide implementation details behind a clean API
```

You can mix both patterns: functions orchestrate flows, classes encapsulate objects.

_____

Mini Task — Task Manager OOP

Inside projects/task_manager/:

1. Create task.py with a Task class containing title, completed, priority="normal", and methods mark_done() plus describe() returning a formatted summary.
2. Create main.py that instantiates several tasks, stores them in a list, and prints each description.
3. Stretch goal: add a TaskList class with add_task(), list_pending() to practice classes working together.

This OOP layer will make it easier to structure file, database, and web projects in upcoming modules.

_____

Thanks!

By ElnurBDa