DAY 16: Object Oriented Programing

11/02/2025

Why do we need OOP and how does it work?

As projects grow, procedural programming can become difficult to manage. Functions start interacting with multiple variables, making the code hard to track and maintain. OOP provides a structured approach by breaking down complex tasks into **smaller, reusable modules**.

Key Concepts

- **Modularization**: Large projects are divided into smaller, manageable components.
- **Scalability**: Easier to expand and maintain code.
- **Reusability**: Once a module is created, it can be used in other projects without rewriting.

Real-World Analogy

Imagine running a restaurant alone—you take orders, cook food, and clean up. This is like procedural programming: everything is managed in one place, leading to inefficiency. With OOP, you have a structured team—chefs, waiters, cleaners—each handling their specific role, making operations smoother.

OOP helps structure code similarly, allowing for better organization, collaboration, and scalability in larger projects.

How to use OOP: Classes and Objects

OOP is all about modeling real-world objects in code. Using the restaurant analogy, we can create a **virtual restaurant** where each role (chef, waiter, cleaner, manager) is represented as an object.

Key Concepts in OOP

1. **Objects** – Represent real-world entities (e.g., a waiter).
2. **Attributes** – The properties an object has (e.g., tables a waiter is responsible for).
3. **Methods** – The actions an object can perform (e.g., taking orders, processing payments).

Classes and Objects

- A **class** is a **blueprint** for creating objects.

- Each object is an **instance** of a class.
- Example: A `Waiter` class can be used to create multiple waiter objects (`Henry`, `Betty`).

OOP allows us to **group data (attributes) and functionality (methods) together**, making the code more structured and reusable.

Constructing Objects and Accessing their Attributes and Methods

What are Classes and Objects?

- A **class** is like a **blueprint** for making objects. Think of it like a car design that defines color, wheels, speed, etc.
- An **object** is an actual **instance** of a class, like a real car built using that design.

Using Python's `turtle` Module

Python comes with a built-in `turtle` module that allows us to create graphics.
We can create and control a turtle that moves on the screen.

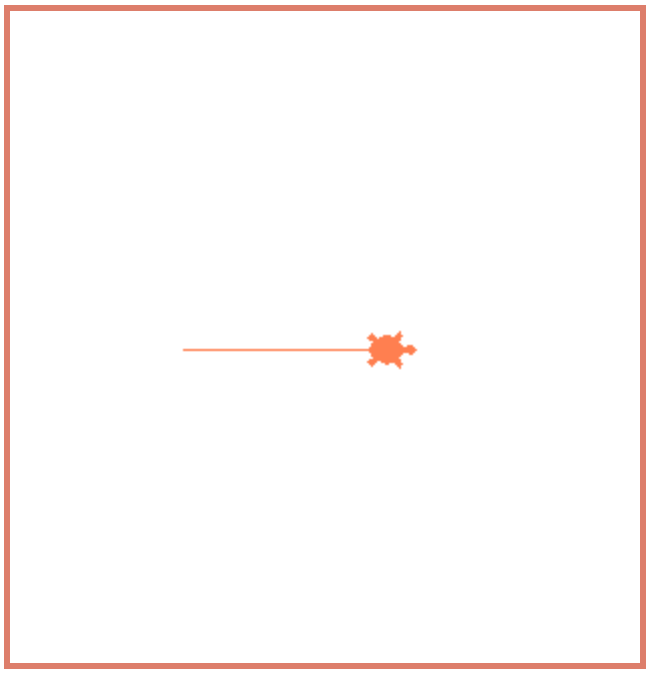
```
from turtle import Turtle, Screen

timmy = Turtle()  # Create a turtle object
my_screen = Screen()  # Create a screen (window) for the turtle
```

Attributes and Methods

- **Attributes** store object information (e.g., screen size).
- **Methods** are actions the object can perform (e.g., move forward).

```
3
4   print(another_module.another_variable)
5   from turtle import Turtle, Screen
6   timmy = Turtle()
7   print(timmy)
8   timmy.shape("turtle")
9   timmy.color("coral")
10  timmy.forward(100)
11  my_screen = Screen()
12  print(my_screen.canvheight)
13  my_screen.exitonclick()
```



How to Add Python Packages and use PyPi

1. What are Packages?

- A **package** is a collection of multiple Python files (modules) written by other developers to achieve a specific goal.
- Unlike built-in modules, packages must be **installed separately** before use.

2. Python Package Index (PyPI)

- PyPI is a repository of Python packages shared by the community.
- Developers can browse, install, and use packages from PyPI in their projects.

Open PyCharm

Go to Preferences/Settings:

- On Windows: Click `File` → `Settings`.
- On macOS: Click `PyCharm` → `Preferences`.

Select the Python Interpreter:

- Navigate to `Project: <Your Project Name>` → `Python Interpreter`.

Install PrettyTable:

- Click the + (plus) button.
- Search for `prettytable` in the available packages.
- Click **Install Package**.

Practice Modifying Object Attributes and Calling Methods

Let's use our example from the turtle to construct our table, now we do it with pretty table that we install previously:

```
"""
from prettytable import PrettyTable
table = PrettyTable()
print(table)

```

It prints this:

```
"C:\Users\Valentina\PycharmProjects\Pyt
++
||
++
++

Process finished with exit code 0
```

Now it's time to construct a prettytable object and practice using its methods and attributes. Our starting point as always is the documentation:
https://pypi.org/project/prettytable/
Use `add_column(field_name, data_list)` to add data.

```
from prettytable import PrettyTable
table = PrettyTable()
table.add_column( fieldname: "Pokemon Name", column: ["Pikachu", "Squirtle", "Charmander"])
table.add_column( fieldname: "Type", column: ["Electric", "Water", "Fire"])
print(table)
```

If we want to change the alignment of our text in the table we can type:

```
table.add_column( fieldname: "Type" , column: [ "El
table.align = "c" #Center aligning text
print(table)
```

Building the Coffee Machine in OOP

- Download the **zip file** from the course resources.
- Extract/unzip the file.
- Open the project folder in **PyCharm**.

coffee_maker.py | main.py | menu.py | money_machine.py

```
from menu import Menu
from coffee_maker import CoffeeMaker
from money_machine import MoneyMachine
menu = Menu()
print(menu.get_items())
money_machine = MoneyMachine()

# Test the functionality
money_machine.report()  # Should print: Money: $0
payment_success = money_machine.make_payment(2.5)  # Simulate buying something for $2.50

if payment_success:
    print("Transaction successful!")
else:
    print("Transaction failed!")


drink = menu.find_drink("espresso")
if drink:
    print(f"Found: {drink.name}, Cost: ${drink.cost}")
    print(f"Ingredients: {drink.ingredients}")
else:
    drink = menu.find_drink("mocha")


coffee_maker = CoffeeMaker()
```

```
# Print available resources
coffee_maker.report()

# Get a drink from the menu
drink = menu.find_drink("espresso")  # Finds the MenuItem object for espresso

# Check if resources are sufficient
if coffee_maker.is_resource_sufficient(drink):
    coffee_maker.make_coffee(drink)
else:
    print("Resources not sufficient.")
```