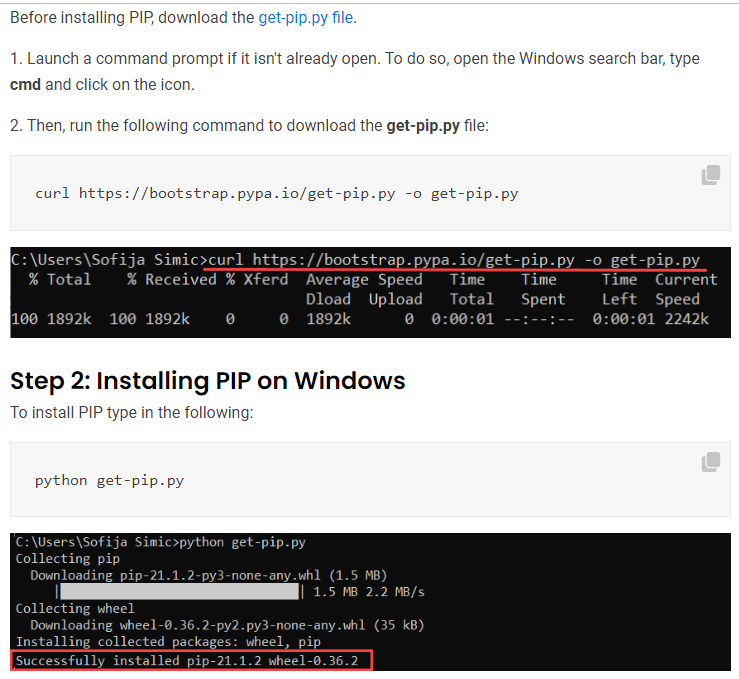
Day - 16

**OOP & Coffee Machine V2**

Object oriented programming and Coffee machine project with OOP

**16.1 Install python-package-installer PIP**



* Also install pylint : pip install pylint

**16.2 Object oriented programming**

|  |  |
| --- | --- |
| Benefits | Idea: It's a abstraction which encapsulates variables (has) and functions (does). |
| * Split a problem * Reusable code * Tasks can be distributed to different classes * Easy to management | * Has: Variables * Does: Functionality/operations. Function |

**16.3 Class naming convention**

Variables are named using '\_' and lowercase. Classes are named using Pascal-case with Camel-case.

**16.4 Declaring Class**

* Python Classes/Objects: Python is an object oriented programming language. Almost everything in Python is an object, with its properties/variables and methods/functions (these are attributes).
* Class: A Class is like an object constructor, or a "blueprint" for creating objects.
* Object: an ***object*** of its ***class*** is a kind of ***variable*** which type is the ***class***.
* Create a Class: To create a class, use the keyword ***class***. Eg: Create a class named ***MyClass***, with a property named ***x***:

**class** MyClass:

  x = 5

* Notice  no  "**()**"
* Creating object:  Create an object named ***a\_obj***, and print the value of ***x***. '**.**' is used to access objects properties.

a\_obj = **MyClass**()

**print**(a\_obj**.**x) #*Accessing class properties in objects*

* Notice  "**()**".  calling constructor ?
* Creating object from MODULES:  object\_name = module\_name.**class\_name()**

**import** turtle

timm = turtle**.Turtle**()

**16.5 The \_\_init\_\_(), self Parameter and Object Methods**

* **\_\_init\_\_():**To understand the meaning of classes we have to understand the built-in **\_\_init\_\_()** function. All classes have a function called **\_\_init\_\_()**, which is always ***executed*** when the ***class*** is being ***initiated***.
* Use the ***\_\_init\_\_()*** function to assign values to object properties, or other operations that are necessary to do when the object is being created.
* The ***\_\_init\_\_()*** function is called *automatically* every time the class is being used to *create* a *new* *object*.

**class** MyClass:

**def** **\_\_init\_\_**(self, a, b):

        #*self referred to the object itself "MyClass"*

**self.**a = a

**self.**b = b

**def** **my\_func**(self):

**print**(f"Yo babe you passed the {**self.**b}")

a\_obj = **MyClass**(3, 4)

a\_obj**.my\_func**()

* Object Methods: Objects can also contain methods. Methods in objects are functions that belong to the object. Eg: notice above **my\_func**()
* **self**: The ***self*** parameter is a reference to the current instance of the class, and is used to ***access*** ***variables*** that ***belong*** to the ***class***.
* It does not have to be named ***self*** , you can call it whatever you like, but it has to be the ***first*** ***parameter*** of any function in the class. Following uses ***tb*** and ***mb*** as first parameter to refer the class itself

**class** MyClass:

**def** **\_\_init\_\_**(**tb**, a, b):

        #*self referred to the object itself "MyClass"*

**tb.**a = a

**tb.**b = b

**def** **my\_func**(**mb**):

**print**(f"Yo babe you passed the {**mb.**b}")

**16.6 Modify, delete, pass**

* Modify Object Properties: You can modify properties on objects like this:

p1.age = 40

* Delete Object Properties: You can delete properties on objects by using the **del** keyword:

**del** p1.age

* Delete Objects: You can delete objects by using the **del** keyword. For example, Delete the p1 object:

del p1

* The pass Statement: ***class*** definitions cannot be empty, but if you for some reason have a class definition with no content, put in the ***pass*** statement to avoid getting an error. Example:

**class** Person:

**pass**

**16.7 Use a Package and its classes: Prettytable**

Use PIP. pip install: **python -m pip install -U prettytable**

Probably is located in **c/appdata/local**. Copy the **.py** file from the downloaded file.

c:\users\rtti\appdata\local\programs\python\python36-32\lib\site-packages (from importlib-metadata->prettytable) (4.0.1)

**import** prettytable

table = prettytable**.PrettyTable**()

#*Adding columns*

table**.add\_column**("Bilie", ["Cry", "Work", "Crap", "Drink", "work", "cry"])

table**.add\_column**("Meesy", ["Laugh", "Work", "Shit", "Play", "Work", "Laugh"])

#*align: Justify text to left*

#*Use "align" attribute. The allowed strings are "l", "r" and "c" for left, right and centre alignment, respectively*

table**.**align = "l"

**print**(table)

#*python oop\_prty\_tabl.py*

**16.8 Coffee Machine Classes Documentation**

|  |  |
| --- | --- |
| **MenuItem Class**  Attributes:     * **name**   (str) The name of the drink.  e.g. “latte”   * **cost**   (float) The price of the drink.  e.g 1.5 |  |

* **ingredients**

(dictionary) The ingredients and amounts required to make the drink.

e.g. {“water”: 100, “coffee”: 16}

**Menu Class**

Methods:

* **get\_items()**

Returns all the names of the available menu items as a concatenated string.

e.g. “latte/espresso/cappuccino”

* **find\_drink(order\_name)**

Parameter order\_name: (str) The name of the drinks order.

Searches the menu for a particular drink by name. Returns a MenuItem object if it exists, otherwise returns None.

**CoffeeMaker Class**

Methods:

* **report()**

Prints a report of all resources.

e.g.

Water: 300ml

Milk: 200ml

Coffee: 100g

* **is\_resource\_sufficient(drink)**

Parameter drink: (MenuItem) The MenuItem object to make.

Returns True when the drink order can be made, False if ingredients are insufficient.

e.g.

True

* **make\_coffee(order)**

Parameter order: (MenuItem) The MenuItem object to make.

Deducts the required ingredients from the resources.

**MoneyMachine Class**

Methods:

* **report()**

Prints the current profit

e.g.

Money: $0

* **make\_payment(cost)**

Parameter cost: (float) The cost of the drink.

Returns True when payment is accepted, or False if insufficient.

e.g. False

* Exercise 16.1 : Coffee Machine project.

Practice version

**import** coffee\_maker

**import** money\_machine

**import** menu

#*alawys use "()" ti initiate class to objects*

drink\_menue = menu**.Menu**()

cofy\_serv = coffee\_maker**.CoffeeMaker**()

pay\_bill = money\_machine**.MoneyMachine**()

machine\_run = **True**

**while** machine\_run **==** **True**:

    ask = **input**(f"what drinks do you want : {drink\_menue**.get\_items**()}")**.lower**()

**if** ask **==** "report":

        #*print(cofy\_serv.report())*

        cofy\_serv**.report**()

**elif** ask **==** "off":

        machine\_run = **False**

**else**:

        picked\_drink = drink\_menue**.find\_drink**(ask)

**print**(f"\n\nYou ordered for \"{picked\_drink**.**name}\", its price is ${picked\_drink**.**cost}")

**print**("\n----------Current Available resource -----------\n")

        #*printing "cofy\_serv.report()" or "pay\_bill.report()"only prints "NONE" its just calling a function/method*

        #*print(cofy\_serv.report())*

        #*print(f"current profit : {pay\_bill.report()}")*

        cofy\_serv**.report**()

        pay\_bill**.report**()

        sufficient\_ingredient = cofy\_serv**.is\_resource\_sufficient**(picked\_drink)

**if** sufficient\_ingredient **==** **True** :

**print**("\n----------Make payment -----------")

            transaction = pay\_bill**.make\_payment**(picked\_drink**.**cost)

**if** transaction **==** **True**:

                cofy\_serv**.make\_coffee**(picked\_drink)

#*python cfy\_mcn\_oop.py*

**class** MenuItem:

    """Models each Menu Item."""

**def** **\_\_init\_\_**(self, name, water, milk, coffee, cost):

**self.**name = name

**self.**cost = cost

**self.**ingredients = {

            "water": water,

            "milk": milk,

            "coffee": coffee

        }

**class** Menu:

    """Models the Menu with drinks."""

**def** **\_\_init\_\_**(self):

**self.**menu = [

**MenuItem**(name="latte", water=200, milk=150, coffee=24, cost=2.5),

**MenuItem**(name="espresso", water=50, milk=0, coffee=18, cost=1.5),

**MenuItem**(name="cappuccino", water=250, milk=50, coffee=24, cost=3),

        ]

**def** **get\_items**(self):

        """Returns all the names of the available menu items"""

        options = ""

**for** item **in** **self.**menu:

            options += f"{item**.**name}/"

**return** options

**def** **find\_drink**(self, order\_name):

        """Searches the menu for a particular drink by name. Returns that item if it exists, otherwise returns None"""

**for** item **in** **self.**menu:

**if** item**.**name **==** order\_name:

**return** item

**print**("Sorry that item is not available.")

**class** CoffeeMaker:

    """Models the machine that makes the coffee"""

**def** **\_\_init\_\_**(self):

**self.**resources = {

            "water": 300,

            "milk": 200,

            "coffee": 100,

        }

**def** **report**(self):

        """Prints a report of all resources."""

**print**(f"Water: {**self.**resources['water']}ml")

**print**(f"Milk: {**self.**resources['milk']}ml")

**print**(f"Coffee: {**self.**resources['coffee']}g")

**def** **is\_resource\_sufficient**(self, drink):

        """Returns True when order can be made, False if ingredients are insufficient."""

        can\_make = **True**

**for** item **in** drink**.**ingredients:

**if** drink**.**ingredients[item] **>** **self.**resources[item]:

**print**(f"Sorry there is not enough {item}.")

                can\_make = **False**

**return** can\_make

**def** **make\_coffee**(self, order):

        """Deducts the required ingredients from the resources."""

**for** item **in** order**.**ingredients:

**self.**resources[item] -= order**.**ingredients[item]

**print**(f"Here is your {order**.**name} !!!!. Enjoy!")

**class** MoneyMachine:

    CURRENCY = "$"

    COIN\_VALUES = {

        "quarters": 0.25,

        "dimes": 0.10,

        "nickles": 0.05,

        "pennies": 0.01

    }

**def** **\_\_init\_\_**(self):

**self.**profit = 0

**self.**money\_received = 0

**def** **report**(self):

        """Prints the current profit"""

**print**(f"Money: {**self.**CURRENCY}{**self.**profit}")

**def** **process\_coins**(self):

        """Returns the total calculated from coins inserted."""

**print**("Please insert coins.")

**for** coin **in** **self.**COIN\_VALUES:

**self.**money\_received += **int**(input(f"How many {coin}?: ")) \* **self.**COIN\_VALUES[coin]

**return** **self.**money\_received

**def** **make\_payment**(self, cost):

        """Returns True when payment is accepted, or False if insufficient."""

**self.process\_coins**()

**if** **self.**money\_received **>=** cost:

            change = **round**(**self.**money\_received - cost, 2)

**print**(f"Here is {**self.**CURRENCY}{change} in change.")

**self.**profit += cost

**return** **True**

**else**:

**print**("Sorry that's not enough money. Money refunded.")

**self.**money\_received = 0

**return** **False**

Instructors solution

**from** menu **import** Menu, MenuItem

**from** coffee\_maker **import** CoffeeMaker

**from** money\_machine **import** MoneyMachine

menu = **Menu**()

coffee\_maker = **CoffeeMaker**()

money\_machine = **MoneyMachine**()

machine\_is\_on = **True**

**while** machine\_is\_on:

    option = menu**.get\_items**()

    choice = **input**(f"What would you like to have? ({option}report/off): ")

**if** choice **==** 'off':

        machine\_is\_on = **False**

**elif** choice **==** 'report':

        coffee\_maker**.report**()

        money\_machine**.report**()

**else**:

        drink = menu**.find\_drink**(choice)

        is\_enough\_resource = coffee\_maker**.is\_resource\_sufficient**(drink)

        is\_payment\_successful = money\_machine**.make\_payment**(drink**.**cost)

**if** is\_enough\_resource **and** is\_payment\_successful:

            coffee\_maker**.make\_coffee**(drink)

#*python main\_git\_done.py*

* Exercise 16.2 : Fun geometry with Turtle-Graphics.

**from** turtle **import** Turtle, Screen

timm = **Turtle**()

**print**(timm)

timm**.shape**("turtle")

timm**.color**("pink")

**for** i **in** **range**(0, 1000):

    timm**.forward**(i)

    timm**.right**(i)

my\_scrn = **Screen**()

**print**(my\_scrn**.**canvheight)

my\_scrn**.exitonclick**()

#*python oop\_intro.py*