Lab: Classes and Objects

This document defines the exercises for the "Python OOP" course at @Software University.

Please submit your solutions (source code) to all the below-described problems in Judge.

1. Vehicle

Create a class called Vehicle. Upon initialization, it should receive **max_speed** (integer, **optional**; **150** by default) and **mileage** (number). Create an instance variable called **gadgets** - an empty list by default.

Examples

Test Code	Output

`car = Vehicle(20)`
`print(car.max_speed)`
`print(car.mileage)`
`print(car.gadgets)`
`car.gadgets.append('Hudly Wireless')`
`print(car.gadgets)`
	`150`
	`20`
	`[]`
	`['Hudly Wireless']`

2. Point

Create a class called **Point**. Upon initialization, it should receive **x** and **y** (**numbers**). Create **3 instance methods**:

- **set_x(new_x)** - **changes the x value** of the point
- **set_y(new_y)** - **changes the y value** of the point
- **__str__()** - returns the **coordinates of the point in the format "The point has coordinates ({x},{y})"**

Examples

Test Code	Output

`p = Point(2, 4)`
`print(p)`
`p.set_x(3)`
`p.set_y(5)`
`print(p)`
	`The point has coordinates (2,4)`
	`The point has coordinates (3,5)`

3. Circle

Create a class called **Circle**. Upon initialization, it should receive a **radius** (**number**). Create a class attribute called **pi** which should be equal to **3.14**. Create **3 instance methods**:

- **set_radius(new_radius)** - changes the **radius**
- **get_area()** - returns the **area of the circle**
- **get_circumference()** - returns the **circumference of the circle**

Examples

Test Code	Output
`circle = Circle(10)` `circle.set_radius(12)`	`452.16` `75.36`

`print(circle.get_area())` `print(circle.get_circumference())`	

4. Glass

Create a class called **`Glass`**. Upon initialization, it will **not receive any parameters**. You must create an **instance attribute** called **`content`** which should be equal to **`0`**. You should also create a **class attribute** called **`capacity`** which should be **`250 ml`**. Create **3 instance methods**:

- **`fill(ml)`** - fills the glass with the given milliliters if there is **enough space** in it and returns **`"Glass filled with {ml} ml"`**, otherwise returns **`"Cannot add {ml} ml"`**
- **`empty()`** - empties the glass and returns **`"Glass is now empty"`**
- **`info()`** - returns info about the glass in the format **`"{space_left} ml left"`**

Examples

Test Code	Output

`glass = Glass()`
`print(glass.fill(100))`
`print(glass.fill(200))`
`print(glass.empty())`
`print(glass.fill(200))`
`print(glass.info())`
	`Glass filled with 100 ml`
	`Cannot add 200 ml`
	`Glass is now empty`
	`Glass filled with 200 ml`
	`50 ml left`

5. Smartphone

Create a class called **`Smartphone`**. Upon initialization, it should receive a **`memory`** (number). It should also have **2 other instance attributes**: **`apps`** (empty list by default) and **`is_on`** (False by default). Create **3 methods**:

- **`power()`** - sets **`is_on`** to **True** if the phone is off, otherwise sets it to **`False`**
- **`install(app, app_memory)`**
  - If there is **enough memory** on the phone and it **is on**, install the app (**add it to apps** and **decrease the memory** of the phone) and return **`"Installing {app}"`**
  - If there **is enough memory**, but the **phone is off**, returns **`"Turn on your phone to install {app}"`**
  - Otherwise, returns **`"Not enough memory to install {app}"`**
- **`status()`** - returns **`"Total apps: {total_apps_count}. Memory left: {memory_left}"`**

Examples

Test Code	Output

`smartphone = Smartphone(100)`
`print(smartphone.install("Facebook", 60))`
`smartphone.power()`
`print(smartphone.install("Facebook", 60))`
`print(smartphone.install("Messenger", 20))`
`print(smartphone.install("Instagram", 40))`
`print(smartphone.status())`
	`Turn on your phone to install Facebook`
	`Installing Facebook`
	`Installing Messenger`
	`Not enough memory to install Instagram`
	`Total apps: 2. Memory left: 20`