# Intro to Computer Science
# CS-UH 1001, Spring 2022

## Lecture 13 – Object Oriented Programming II

# Recap

- Classes are the definition of objects (blueprints)
  - Objects have attributes and methods
- Class definitions are used to instantiate an object
  - Each object will be an instance of the class
- *self* is a reference to the object instance
- Attributes can be private or public
  - getter and setter methods are used to access private attributes

# Recap: Class example

```
class Coin:
    def __init__(self):
        self.__side_up = 'Head'

    def toss(self):
        if randint(0,1) == 0:
            self.__side_up = 'Heads'
        else:
            self.__side_up = 'Tails'

    def get_side_up(self):
        return self.__side_up
```

Special method called Initializer:
Get executed immediately when you instantiate an object.
Used to define and initialize the class attributes

Two underscores define a private attribute that can not be accessed from the outside

Getter method, returns the value of a private attribute to the outside

# Attributes vs. Variables

```
class Coin:
    def __init__(self):
        self.side_up = 'Heads'

    def set_value(self, val):
        self.value = val
        value = val * 2

c = Coin()
print(c.value)
c.set_value(100)
print(c.value)
```

Attribute

Local method variable

AttributeError: 'Coin' object has no attribute 'value'

100

# List of Objects

- Recall: Lists can hold any kind of data type
  - In Python, everything is an object

- Creating a list of objects:

```
coin_list = []
coin = Coin()
coin_list.append(coin)
coin_list.append(Coin())
```

# Looping over Lists of Objects

- Recall: A for loop iterates over all elements in a sequence

- Iterating of a list of objects:

```
for element in coin_list:
    element.toss()
    face = element.get_side_up()
```

element is a Coin object

# Instantiating a Class with Initial Values

- An object can be instantiated with initial values
- Initial values can be added as arguments to the constructor

```
class Coin():
    def __init__(self, color, value):
        self.color = color
        self.value = value
```

Instantiate the object with initial values:

```
coin = Coin("Silver", 2)
coin1 = Coin("Copper", 10)
```

Remember: Arguments can also have default arguments

# __init__ Method with Default Arguments

- We know that the __init__ method defines and initialize the object's attributes

- We can also define how the objects attribute are initialized by:

```
def __init__ (self, face='Heads'):
    self.side_up = face

coin1 = Coin()
coin2 = Coin('Tails')

print(coin1.side_up, coin2.side_up)
```

Default value in case no arguments are given when creating the object

- How about print(coin1)?

  <__main__.Coin object at 0x10b96afd0>

# Special Methods in Classes

- We have already seen the __init__() method

- All Python objects have multiple built-in methods
  - all have a default behavior when invoked on objects
  - but, their behavior can be overwritten (e.g. overloaded operator)

- For example: Printing an object's state
  - A state is the value of the object's attributes at a given moment
  - print(object) prints only the reference
  - For example:
    print(my_list) [1, 2, 3, 4, 5]

# String Representation of Objects

- The __str__() method is a special method that can be overwritten/redefined
  - It returns the string representation of an object
  - The method is called when print() or str() function is invoked on an object

```
print(object)
str(object)
```

# The __str__() Method

- For example, for our Coin class, the __str__() method becomes:

```
def __str__(self):
    return "Side up is: " + self.__side_up

print(coin1)
str(coin1)
```

Note: The method must return a string!

**Breakout session I:**
Point Class

# Point class (ex_13.1.py)

## Create a Point class with two attributes

- x coordinate
- y coordinate

## and two methods for

- printing the x and y coordinates of the object in a readable format, for example: (4,3)
- calculating the distance of a point from the origin
  distance = (x**2 + y**2)**0.5


>>> from
coffee
import *

# Calculating the Midpoint as a Function

```
def midpoint(p1, p2):
    mx = (p1.x + p2.x)/2
    my = (p1.y + p2.y)/2
    return Point(mx, my) # return a new Point object

p1 = Point(2, 4)
p2 = Point(1, 5)
p3 = midpoint(p1, p2)
print(p3)
```

(1.5, 4.5)

Note that objects can be sent as arguments to a function.
Remember: objects are mutable

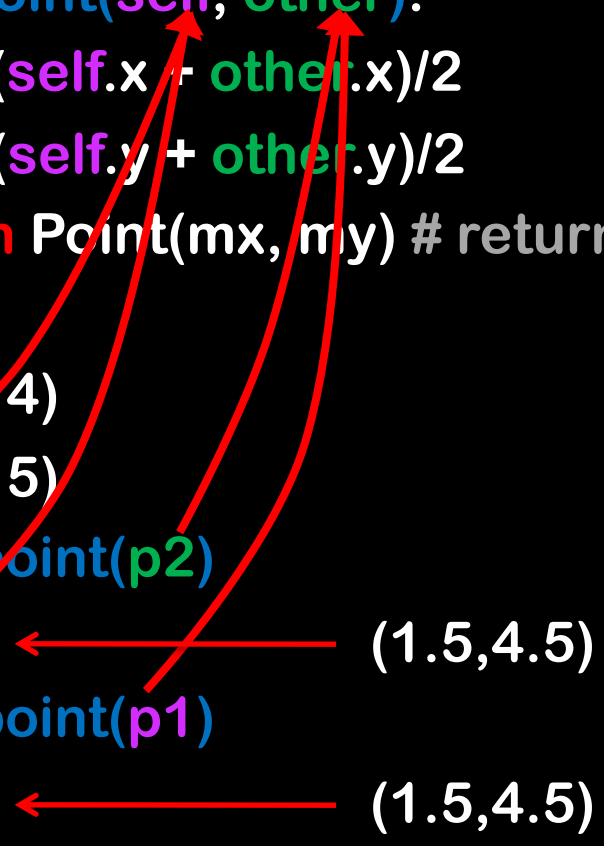
Create a new Point object and return it

How to implement this as a method?

# Calculating the Midpoint as a Method

```
class Point:
    .
    .
    def midpoint(self, other):
        mx = (self.x + other.x)/2
        my = (self.y + other.y)/2
        return Point(mx, my) # return a new Point object

p1 = Point(2, 4)
p2 = Point(1, 5)
p4 = p1.midpoint(p2)
print(p4)
p5 = p2.midpoint(p1)
print(p5)
```



(1.5,4.5)

(1.5,4.5)

**Breakout session II:**
Point Class

# Point class (ex_13.1.py)

Create a point class with the attributes

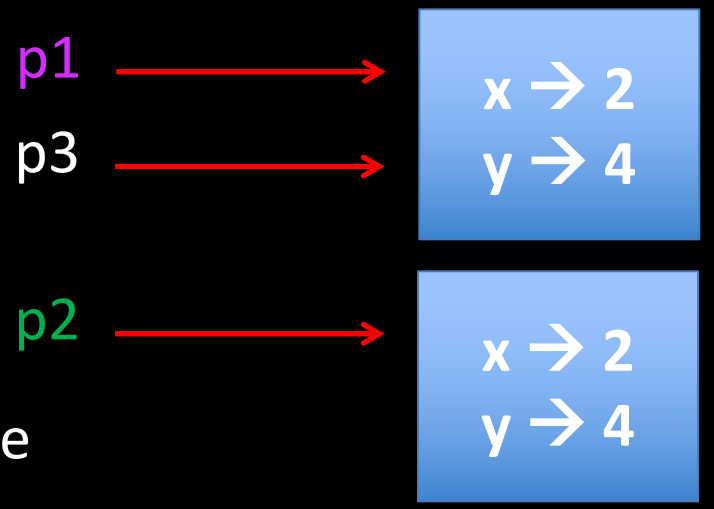
- x coordinate
- y coordinate

and methods for:

- printing the coordinates of the class in a readable format (e.g. (4,3))
- calculating the distance of a point from the origin
  (x**2 + y**2) **0.5
- calculating the midpoint of two points
  xm = (x1 + x2)/2
  ym = (y1 + y2)/2
- calculating the distance to another point
  ((x1-x2)**2 + (y1-y2)**2)**0.5

# Comparison of Objects

```
p1 = Point(2, 4)
p2 = Point(2, 4)
```




```
print(p1 == p2)
```

False

```
p3 = p1
print(p3 == p1)
```

True

The comparison operator does not check if the Point coordinates are the same, it checks if the memory locations (references) of the objects are the same!

# Comparisons of Objects

- Check if the points (x, y) coordinates are the same, you can implement a method:

```
class Point:

    # method to compare the x and y attributes
    def same_coordinates(self, other):
        return (self.x == other.x) and (self.y == other.y)


p1 = Point(2, 4)
p2 = Point(2, 4)
print(p1.same_coordinates(p2))
```

True

# Object Comparison

- In Python you can also define your own comparison methods and overload the built-in ones (==, <=, >=, !=, <, >)

- You can do this by overloading the following methods:

```
def __eq__(self, other):    → equal ==
def __le__(self, other):    → less than or equal <=
def __ge__(self, other):    → greater than or equal >=
def __lt__(self, other):→ less than <
def __gt__(self, other):    → greater than >
def __ne__(self, other):    → not equal !=
```

# Comparisons in Objects

- Check if the points (x,y) coordinates are the same using == operator, you can use:

```
def __eq__(self, other):
    return (self.x == other.x) and (self.y == other.y)

p1 = Point(2, 4)
p2 = Point(2, 4)
if p1 == p2:
    print("Both points have the same coordinates")
```

# Other Operators

- Do you remember overloaded operators?
  - '2'+'2' vs. 2+2

- What about objects?

```
p1 = Point(2, 4)
p2 = Point(2, 4)
p3 = p1 + p2
```

At the moment this throws an Error, but the '+' operator can be changed to add two points

# Objects Overloaded Operators

- Adding two 2D points:
  - The result is a new point with its x coordinate being p1.x + p2.x and y coordinate being p1.y + p2.y

- Similarly to the __str__() or __eq__() method, the (+) operator can be overloaded:

```
def __add__(self, other):
    return Point(self.x + other.x, self.y + other.y)

p1 = Point(2, 4)
p2 = Point(2, 4)
p3 = p1 + p2
print(p3)
```

(4,8)

# Objects Overloaded Operators

- Similarly:

- Subtraction:

```
def __sub__(self, other):
```

→ p1-p2

- Multiplication:

```
def __mul__(self, other):
```

→ p1*p2
(two objects)

```
def __rmul__(self, other):
```

→ p1*2
(object*scalar)

Call center
24 hrs (toll-free)
800-5663
٨٠٠٥٦٦٣
مركز الإتصال
٢٤ ساعة (الرقم المجاني)
KONE

Elevator No: 11327470
رقم المصعد

## Safety Instructions:
## إرشادات السلامة:

In case of emergency, press the alarm button and wait for help. Do not attempt to get out on your own.

في حالة الطوارئ , يرجى الضغط على زر الإنذار و إنتظار المساعدة. الرجاء عدم محاولة الخروج بمفردك.

For your safety please stand clear of the doors - keep fingers, clothes and other belongings away from the doors.

حفاظاً على سلامتك الرجاء الإبتعاد عن أبواب المصعد وإبقاء أصابع يدك وملابسك و متعلقاتك الشخصيه بعيدة عن الأبواب.

Children should be accompanied & observed at all times.

ينبغي مرافقة ومراقبة الأطفال في جميع الأوقات أثناء استخدام المصعد.

Do not use the lift in case of fire or earthquake.

الرجاء عدم إستعمال المصعد في حالة الحريق أو الزلازل.

Do not overload the elevator.

رجاء عدم تجاوز حمولة المصعد (الوزن أو العدد المذكور أعلاه).

It is prohibited to smoke inside the elevator.

ع قطعياً التدخين داخل المصعد بموجب القانون.

**Breakout session III:**
Rectangle Class

# Rectangle class (ex_13.1.py)

**Now that we have created a point class, let's create a rectangle class. A rectangle is represented by its upper-left corner point and its size (width and height).**

- Create the initializer method
  - Use the Point class to store the corner point (upper left) of the rectangle
  - Use two attributes to represent the width and the height
- Create the __str__ method (should display ((x,y), width, height) )
- Create a method to resize the rectangle by a delta width and height
- Create a method that moves the rectangle coordinates (origin) by a delta x and delta y

Now, test the following:

```
r1 = Rectangle(10, 5, 100, 50)
print(r1)
r1.resize(25, -10)
print(r1)
r1.move(-10,10)
print(r1)
```