Module 5. Lesson 5. OOP. Inheritance

Checking qualifications

To get started on the work tasks, you first need to demonstrate your knowledge .

Prove that you are ready for brainstorming!

Checking qualifications

What is a class?

What is an instance of a class?

Checking qualifications

A class

- is a common name for lots of objects;
- in programming is a general description of how these objects should be structured.

IT'S a car

An object

Object class

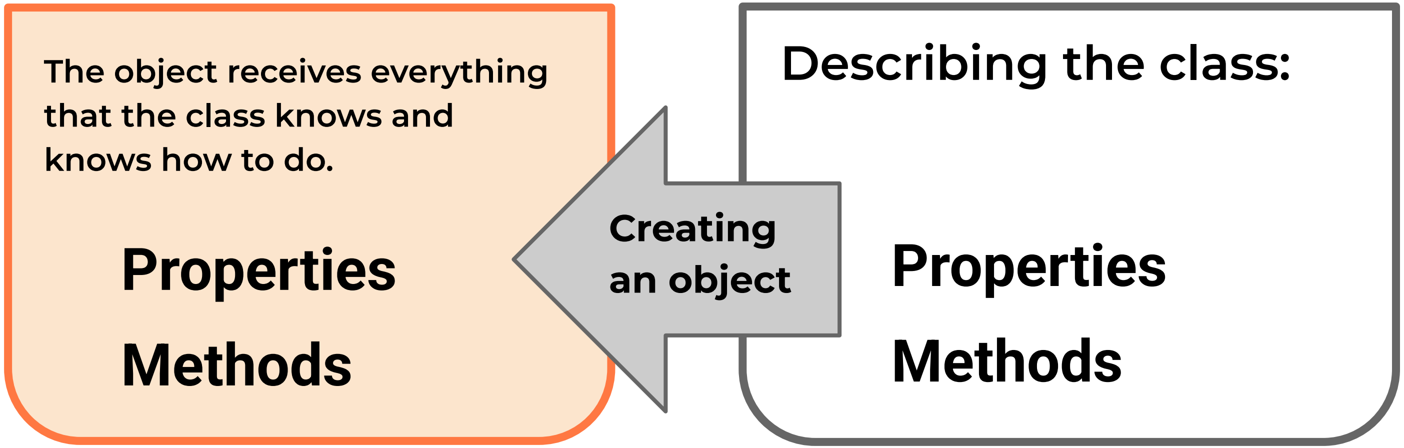
Knowledge about all these objects

Checking qualifications

An instance of a class

is an object created according to the description programmed in the class.

```
instance = Class()
```



The object receives everything that the class knows and knows how to do.

Properties

Methods

Creating an object

Describing the class:

Properties

Methods

Checking qualifications

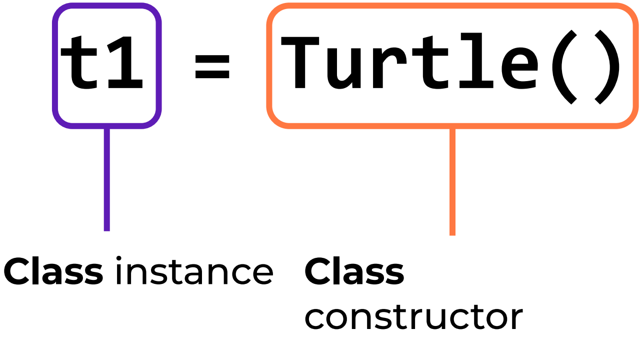
Which Python standard library class do you already know?

How do we create an instance of it?

Checking qualifications

It's the Turtle class

Let's look at creating an *instance* of the Turtle class:



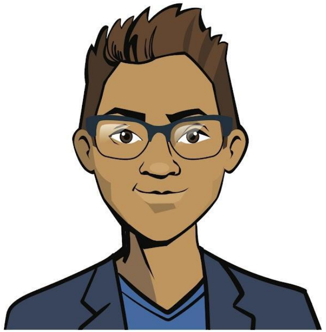
```
t1 = Turtle()
```

Class instance

Class constructor

The class name in parentheses is the **command** that creates a new object of that class.

The result is a link leading to the object (stored in a variable).



Checking qualifications

What is a **class constructor** ?

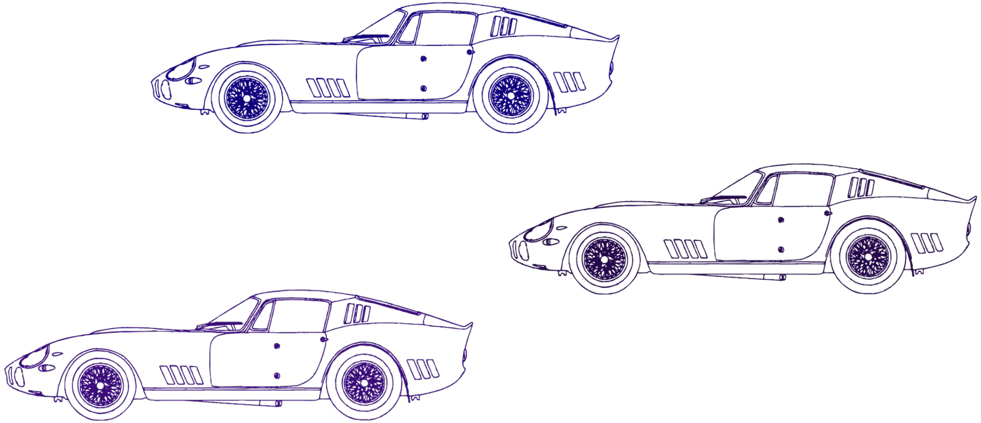
Does it have a universal name?

Checking qualifications

A constructor

is a method that is automatically called when an object is created. It creates an instance of the class.

`def __init__(self,` parameters `)` — constructor name.



Checking qualifications

How do we create our own class?

Where are the properties of the class described?

How do we describe a class method?

Checking qualifications

Creating a class

class is the command that creates a class.

self — current class object.

```
class  Class name  ():

    def __init__(self,  Data  ):
        self. Property  =  Data


    def print_info(self):
        print('Information about the object:', self. Property  )

Instance  =  Class name  ( Property  )
```

A constructor
with the process of
creating an instance of
the class.

Checking qualifications

Qualifications checked!

Great, you are ready to brainstorm and complete your work task!

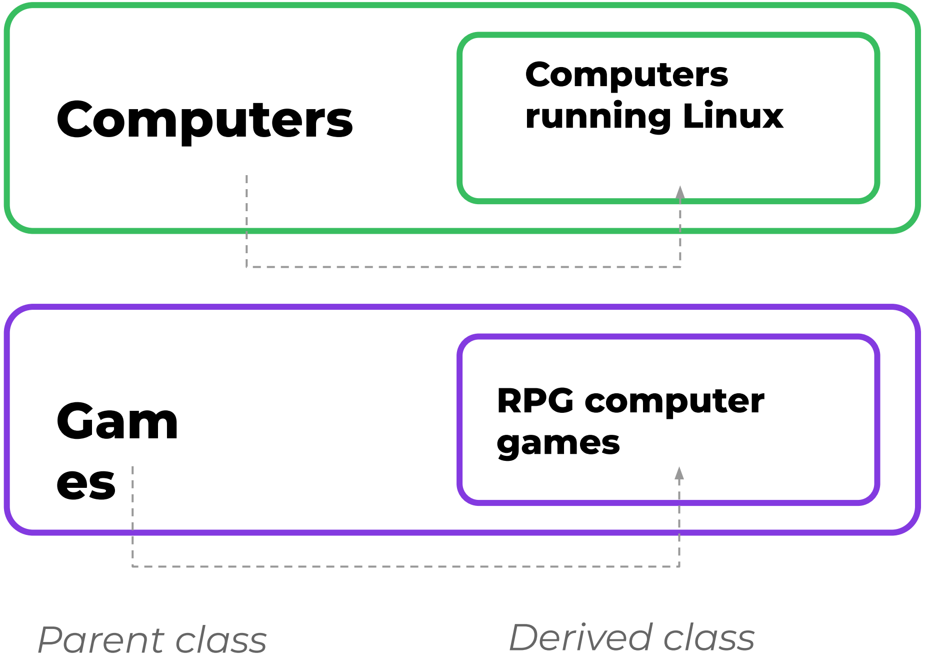
Checking qualifications

Brainstorming:

Inheritance

Classes and subclasses

Think of real-life examples of classes and subclasses.



Computers

Computers running Linux

Gam
es

RPG computer games

Parent class

Derived class

Brain storming

Classes and subclasses

Almost all classes are parents of some classes and derived classes of others.

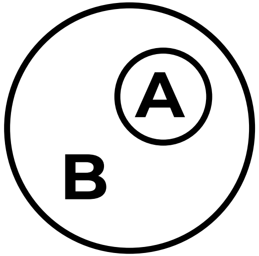
All *computer games* are *programs*

All cats are animals

All desks are tables

All comets are celestial bodies

All automobiles are modes of transportation



Brain storming

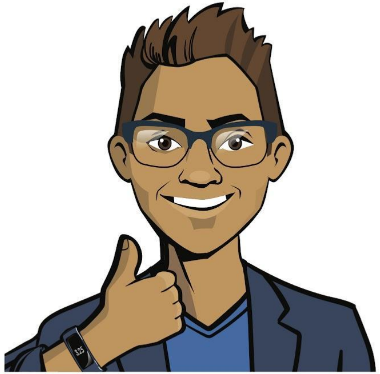
Classes and subclasses

This is very convenient in practice. Example:

1) When you call an **animal** a *cat*, you do not need to specify that it does not fly (most animals cannot fly).

2) When you call a **drink** *tea*, you do not need to specify that it is liquid and that it can quench your thirst.

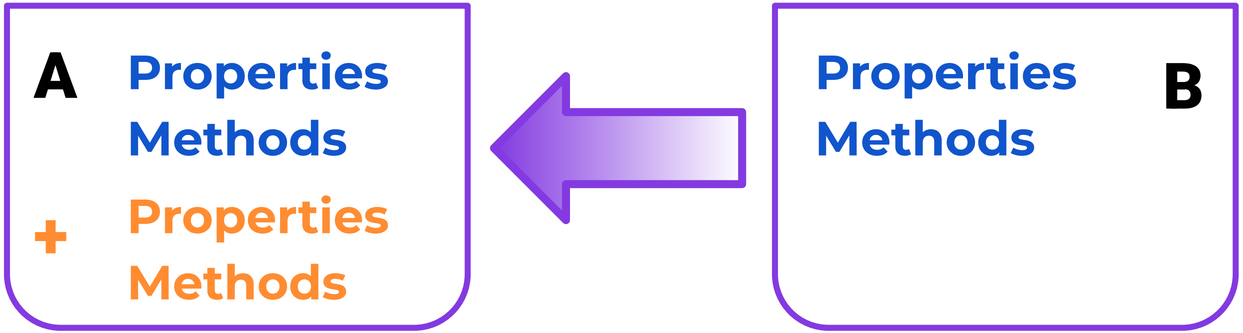
In programming, using information from previously described classes is no less convenient!



Brain storming

Inheritance

Class inheritance helps **transfer all the skills** previously written for a **more general class** into another, more private class, **the inheriting class** .



A Properties

Methods

+ Properties

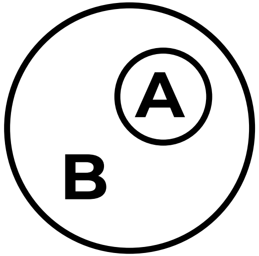
Methods

Properties B

Methods

Derived class

Superclass



Class A is nested within class B

Brain storming

Types of inheritance

Type	*Comment*
The derived class is supplemented with **new methods,** but new properties are not introduced.	No new constructor needed, you can use the superclass constructor!
The derived class is supplemented with **new properties** and **new methods.**	The superclass constructor needs to be supplemented with new properties.

Brain storming

Creating a derived class

Supposing the superclass has already been written, then to create a derived class we need to:

- when creating a derived class, specify the *name of the superclass*;
- add the necessary methods to the derived class.

Method 1:

```
class Derived class name ( Superclass name ):
    def Method name (self, Value ):
        Action with the object and properties
    def Method name (self, Value ):
        Action with the object and properties
```

Option with the introduction of ***only new methods.***

When creating an instance of a derived class, the superclass constructor will be called!

Brain storming

Creating a derived class

Supposing the superclass has already been written, then to create a derived class we need to:

- when creating a derived class, specify the *name of the superclass*;
- add the necessary methods to the derived class.

Method 2:

```
class Derived class name (Superclass name):
    def __init__(self, Value ):
        super().__init__( Value )
    def Method name (self, Value ):
        Action with the object and properties
```

super() calls the superclass to inherit all the properties and methods.

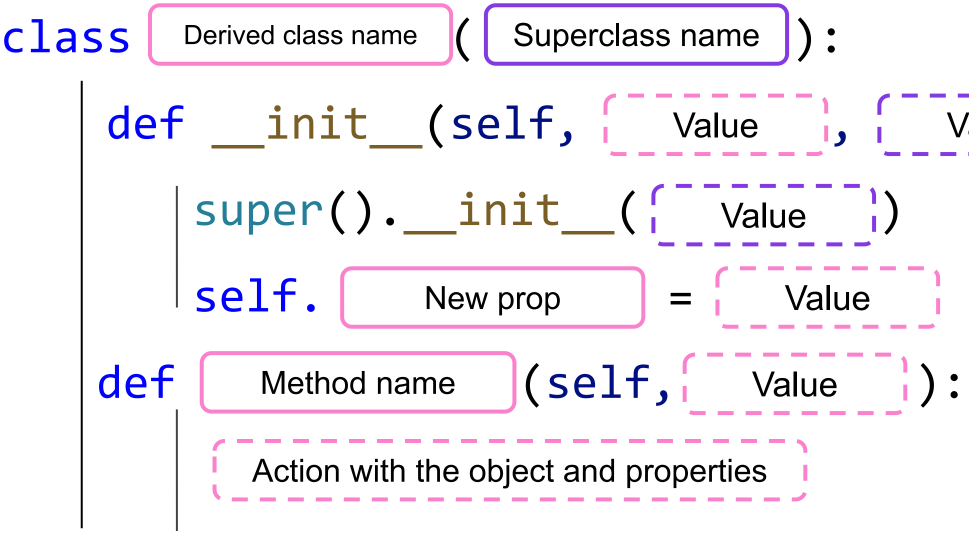
In fact, when creating an instance of a derived class, the superclass constructor is called.

Brain storming

Creating a derived class

To create a derived class we need to:

- when creating a derived class, specify the *name of the superclass*;
- create a constructor, introduce the superclass properties, and add new ones;
- add the necessary methods to the derived class.



```
class Derived class name( Superclass name ):
    def __init__(self, Value , Value ):
        super().__init__( Value )
        self. New prop = Value
    def Method name (self, Value ):
        Action with the object and properties
```

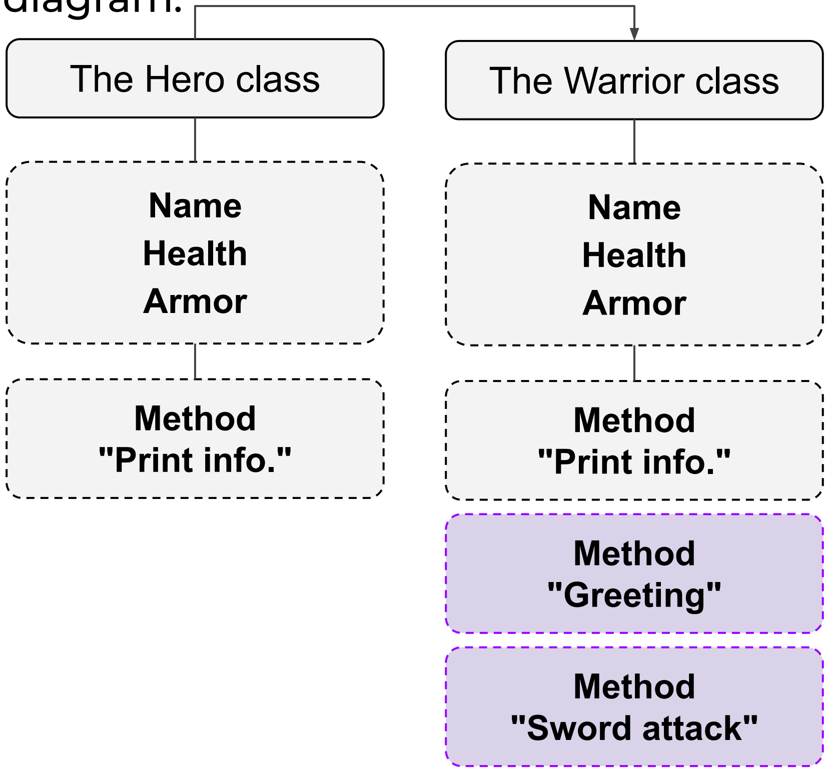
Option ***with the introduction of a new property.***

The constructor takes over the properties of the superclass and adds a new one.

Let's look at a practice task

There is a snippet of code with the Hero class.

The task: to implement the Warrior derived class according to this diagram.



The Hero class

Name
Health
Armor

Method
"Print info."

The Warrior class

Name
Health
Armor

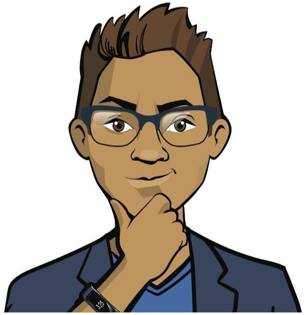
Method
"Print info."

Method
"Greeting"

Method
"Sword attack"

Do we need to introduce new properties? Methods?

How can inheritance be implemented?



Brain storming

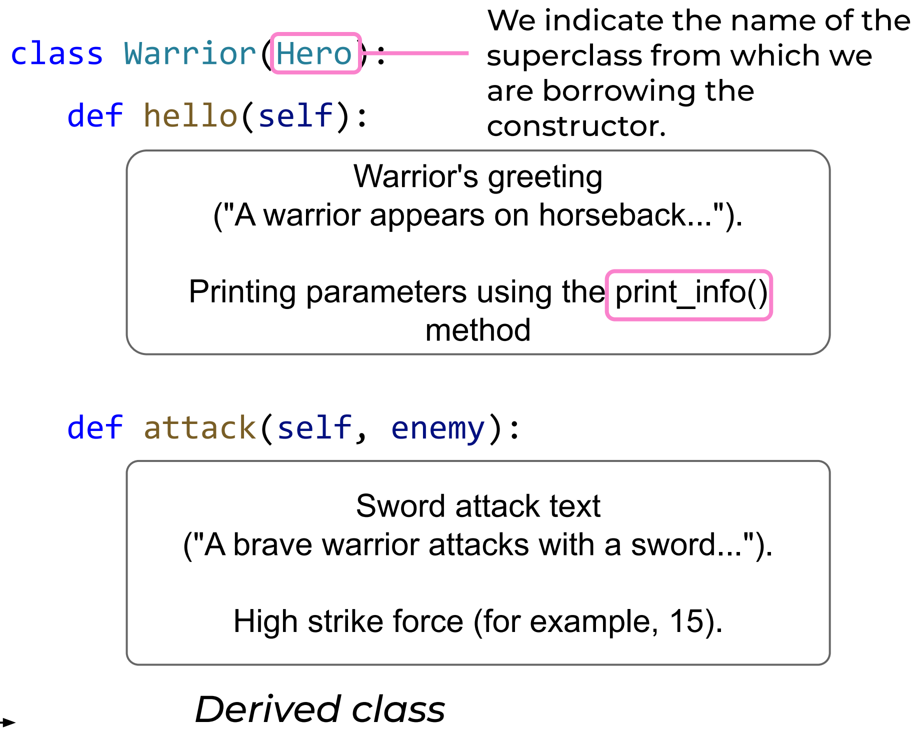
Let's look at a practice task

There is a snippet of code with the Hero class.

The task: to implement the Warrior derived class according to this diagram.

```
class Hero():
   #class constructor
   def __init__(self, name, health, armor):
       self.name = name #string
       self.health = health #number
       self.armor = armor #number
   #print character parameters:
   def print_info(self):
       print('Health level:', self.health)
       print('Armor class:', self.armor, '\n')
```

Superclass →



```
class Warrior(Hero):
   def hello(self):
```

We indicate the name of the superclass from which we are borrowing the constructor.

Warrior's greeting ("A warrior appears on horseback...").

Printing parameters using the print_info() method

```
   def attack(self, enemy):
```

Sword attack text ("A brave warrior attacks with a sword...").

High strike force (for example, 15).

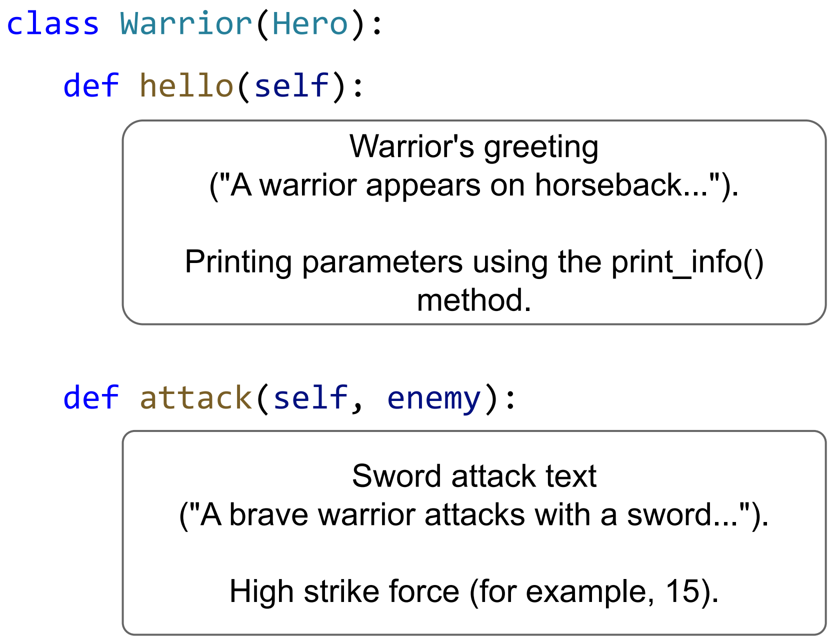
Derived class

Let's look at a practice task

There is a snippet of code with the Hero class.

The task: to implement the Warrior derived class according to this diagram.

```
warrior1 = Warrior('Henry', 100, 50)

warrior1.hello()

warrior1.attack(<Enemy name>)
```



```
class Warrior(Hero):

    def hello(self):
```

Warrior's greeting
("A warrior appears on horseback...").

Printing parameters using the print_info() method.

```
    def attack(self, enemy):
```

Sword attack text
("A brave warrior attacks with a sword...").

High strike force (for example, 15).

Done! You can create an instance of the warrior class and fight the enemy!

The complete code for the Warrior class

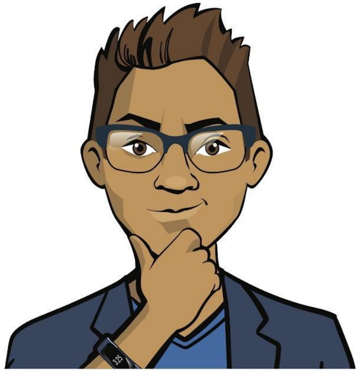
```
class Warrior(Hero):
    def hello(self):
        print('-> NEW HERO. A brave warrior appears riding a horse who is named', self.name)
        self.print_info()
        sleep(4)

    def attack(self, enemy):
        print('-> HIT! A brave warrior', self.name, 'is attacking', enemy.name, 'by sword!')
        enemy.armor -= 15 #strike force for the Warrior class
        if enemy.armor < 0:
            enemy.health += enemy.armor
            enemy.armor = 0
        print('A terrible blow fell upon the enemy. \nNow it's armor: ' +
            str(enemy.armor) + ', and health level: ' + str(enemy.health) + '\n')
        sleep(5)
```

You can use any printing method. The most important things is to keep an eye on the placement of spaces.

In the option with commas, the interpreter automatically separates arguments with spaces.

Tasks:

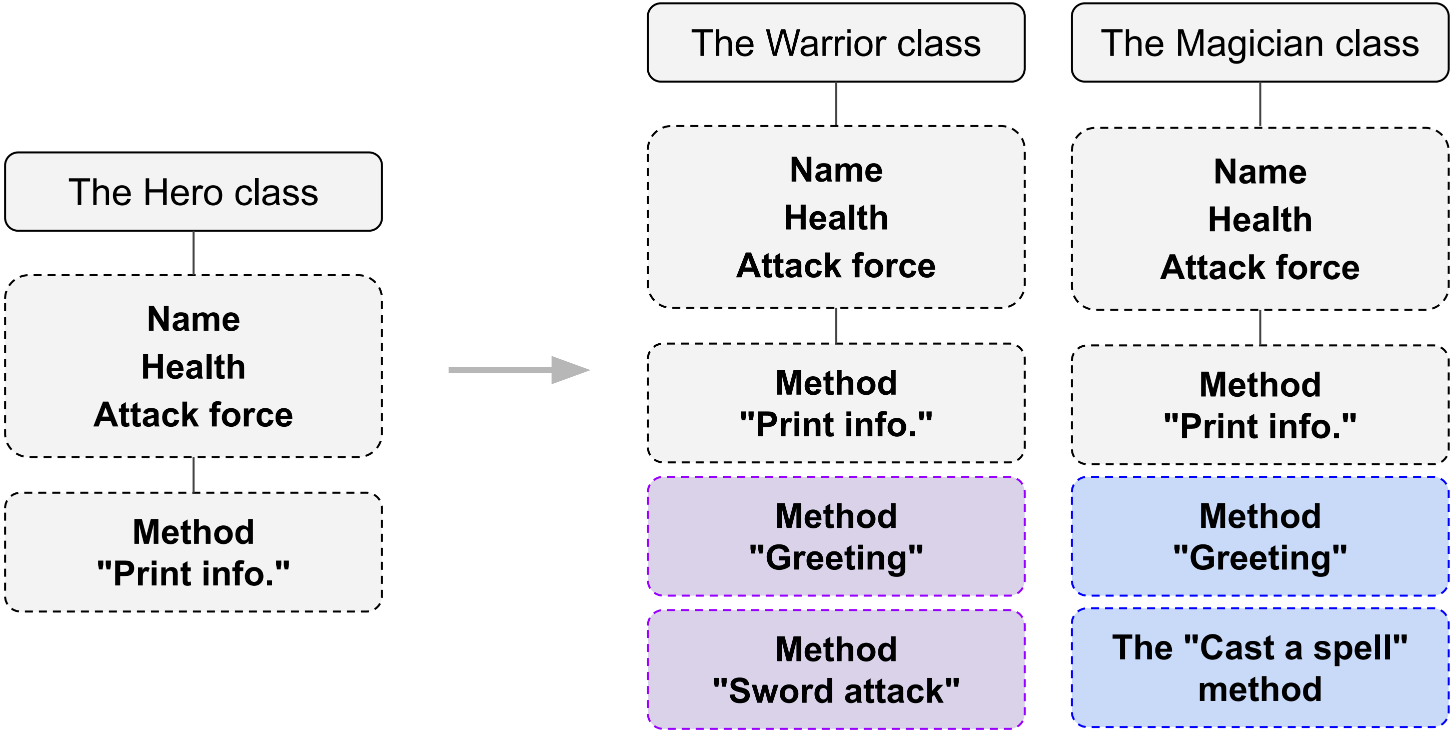
- Create Warrior and Magician derived classes of the Hero superclass (reference class diagrams are available on the platform).
- Create two instances of the class: warrior and magician. Print information about them.
- Program a knight attack on the magician. Then make a counter attack.



Brain storming

Relationship between the Warrior and Magician classes

Use the diagram if needed:



<code>

Working on the platform

Module 5. Lesson 5. OOP. Inheritance

Brainstorming:

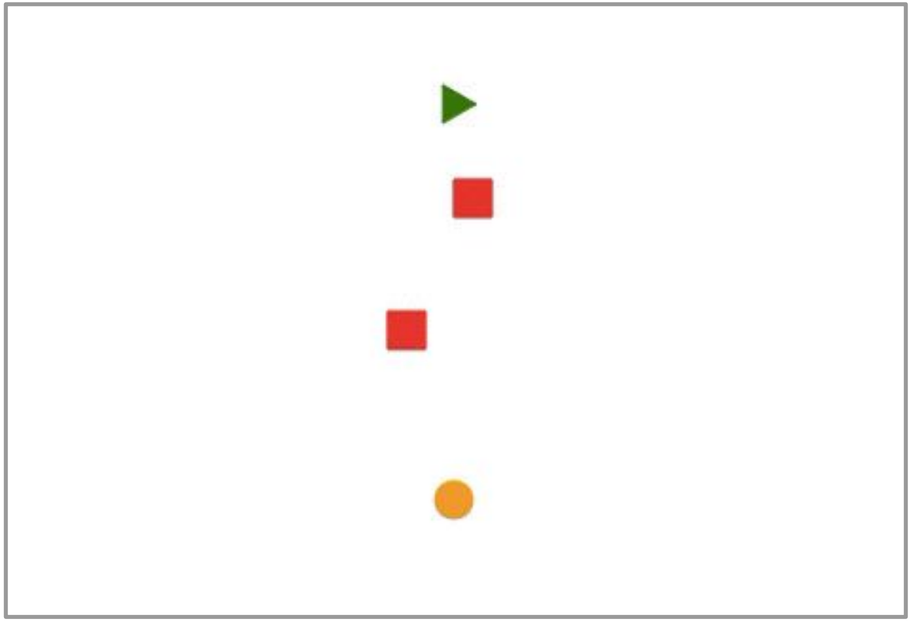
The "Hit It!" game

Programming the "Hit It!" game

As part of the training on creating our own objects, we will program the interactive "Hit It!" game.

The **object of the game** is to get around obstacles and catch the target object.

Expected look of the game:



Brain storming

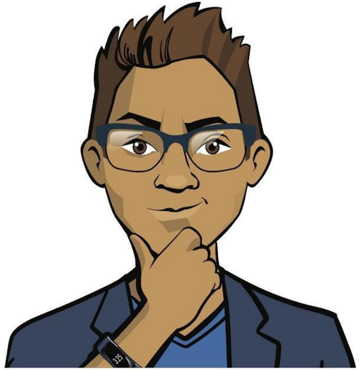
Terms of reference

The **object of the game** is to get around obstacles and catch the target object.

Requirements for the game:

1. **The main object** is keyboard-controlled by the user.
2. At least **two objects** move around the screen automatically and make it difficult to achieve the goal.
3. Victory condition: the player touches the **target object**.
   Then the obstacles disappear.

   Loss condition: the player touches an **obstacle**.
   Then the target disappears.



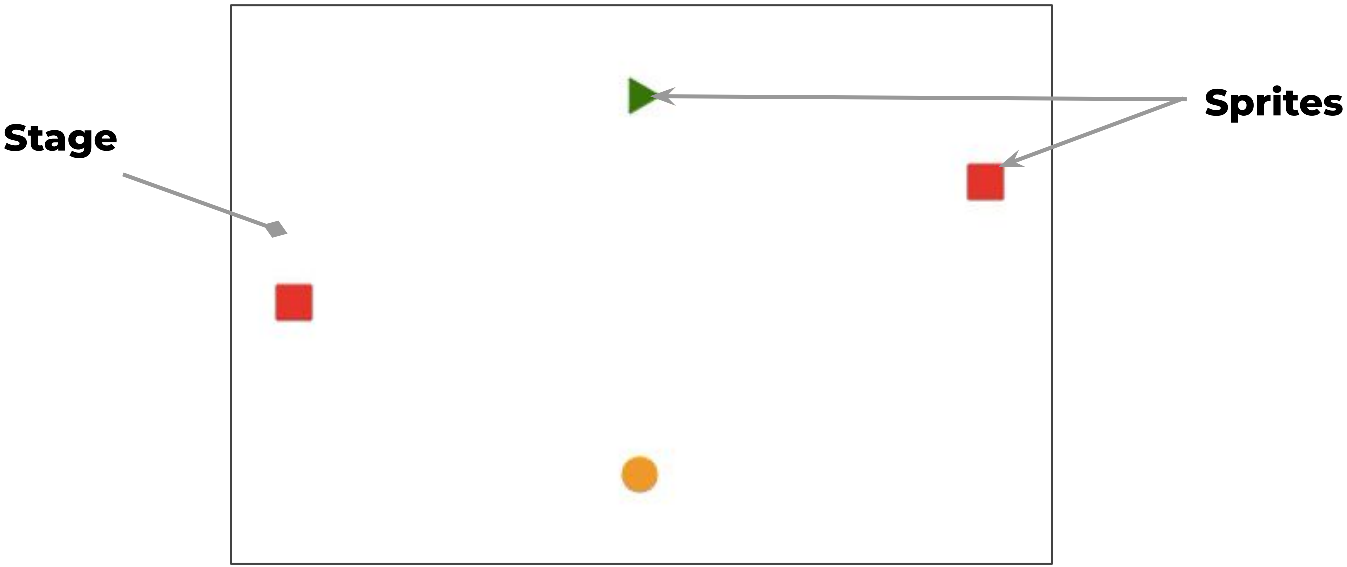
Brain
storming

Terms for creating games

When creating games, developers often use the following terms:

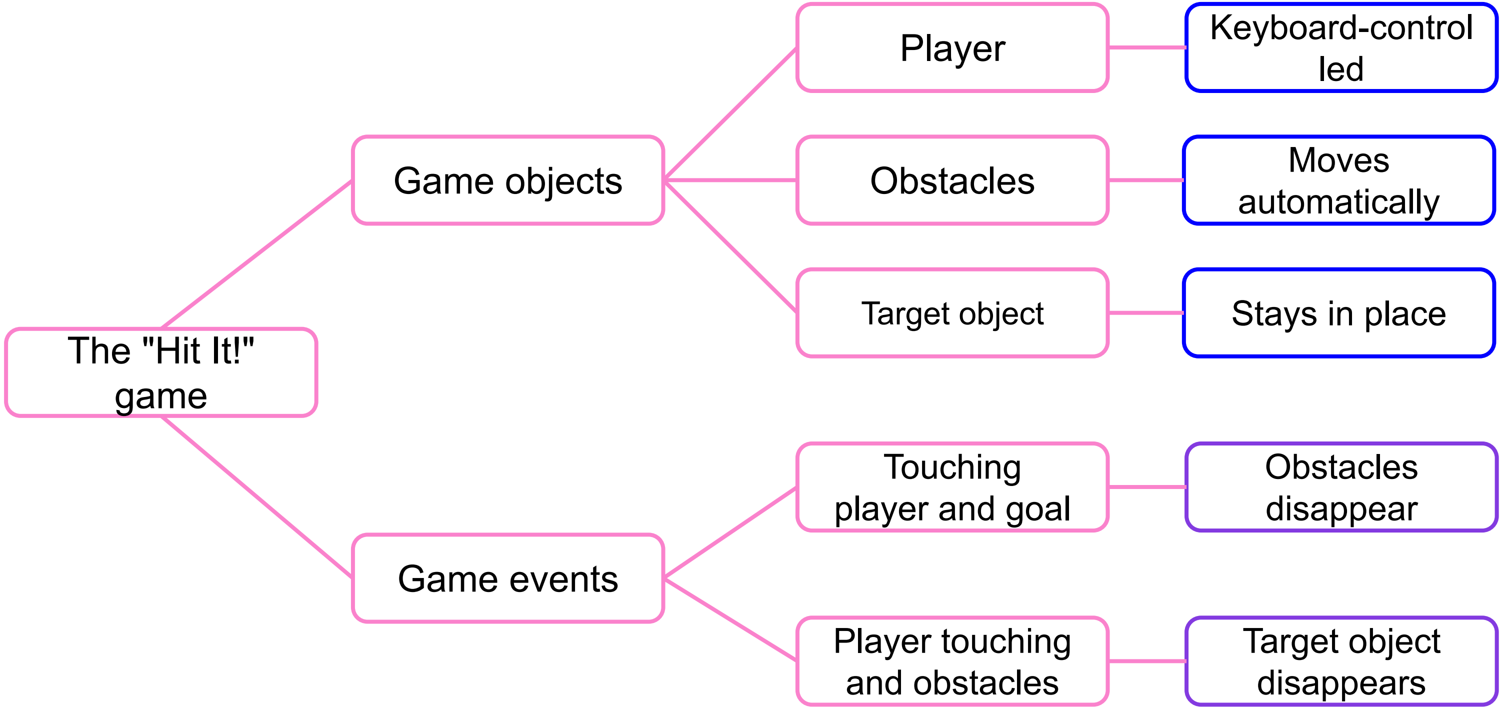
The stage is the "background" of the game. What all the objects move on.

Sprites are any game objects other than the stage.



Planning our work on the project

Let's depict the project's functionality using a mind map:



The "Hit It!" game

Game objects

Player

Keyboard-control led

Obstacles

Moves automatically

Target object

Stays in place

Game events

Touching player and goal

Obstacles disappear

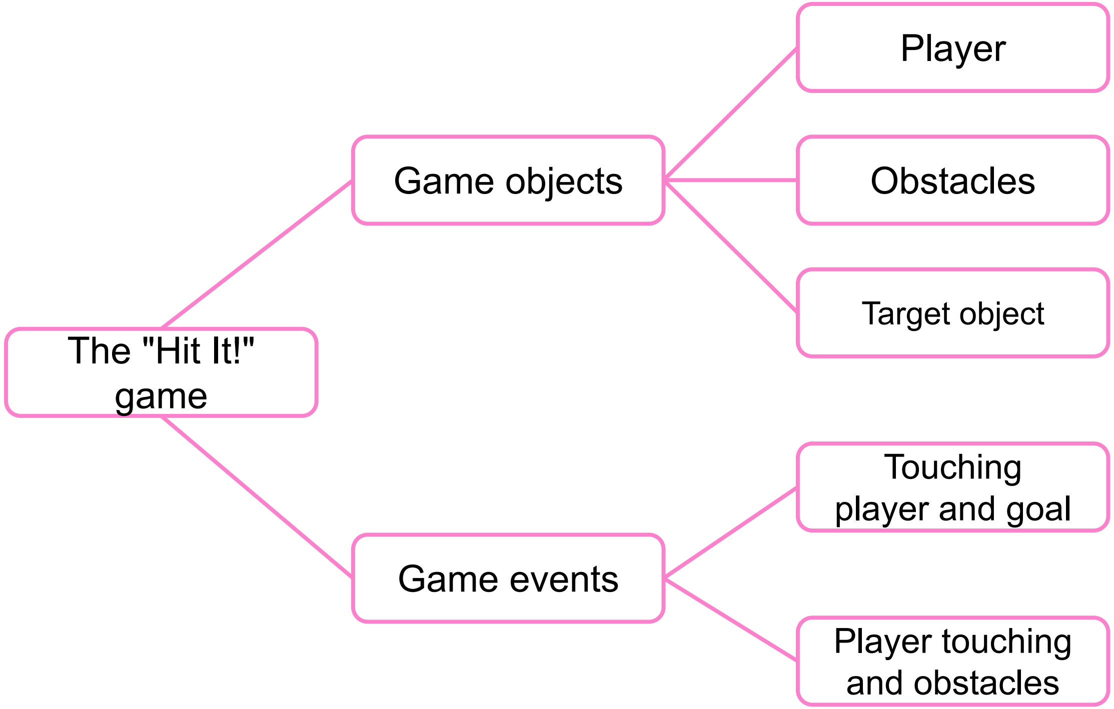
Player touching and obstacles

Target object disappears

Brain storming

Planning our work on the project

Let's depict the project's functionality using a mind map:



There are properties and methods for displaying and moving <u>in the Turtle class!</u>

Not in the Turtle class — this is specific to our game.

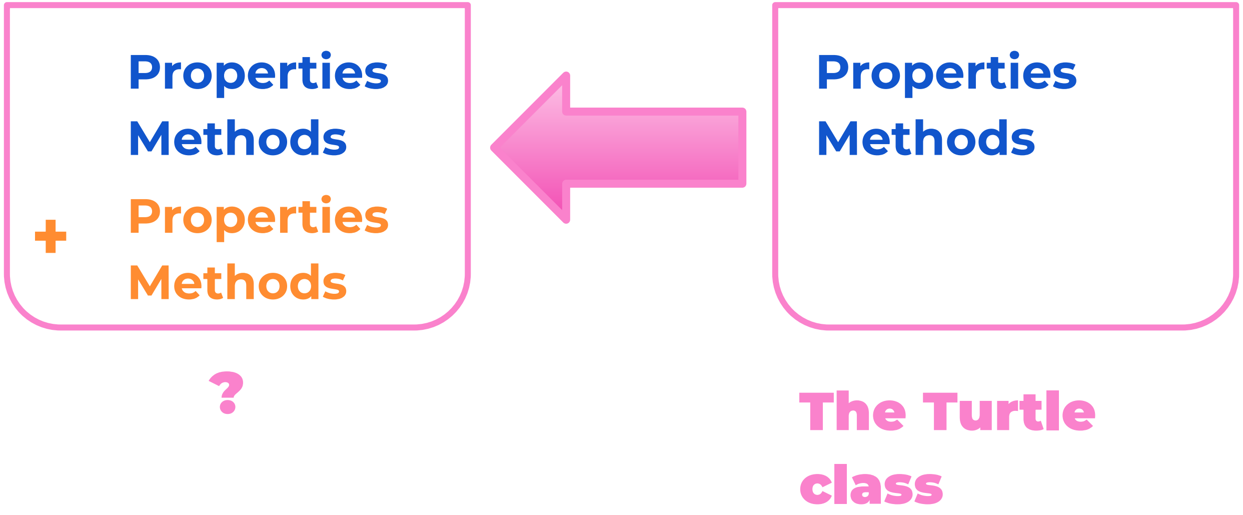
How do we “teach” objects to recognize touching?

Brain storming

Planning our work on the project

If instances of the Turtle class could recognize touching each other, then all sprites could be created by turtles.

But our game also requires handling sprite touches.



Properties
Methods

+ Properties
Methods

?

Properties
Methods

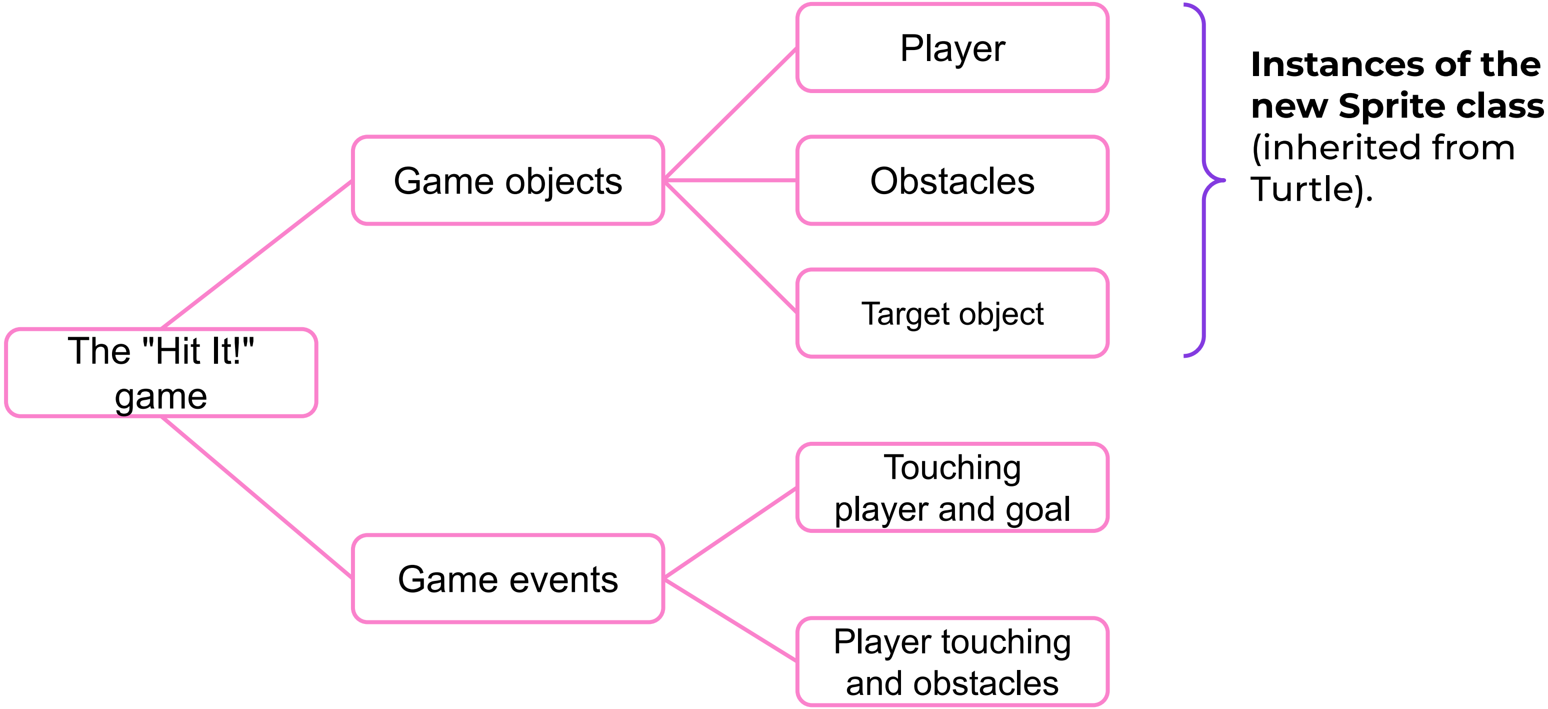
The Turtle class

Can we add the missing methods to the existing Turtle toolkit? How?

Brain storming

Planning our work on the project

Let's create a Sprite derived class from the existing Turtle class and supplement it with the necessary properties and methods.

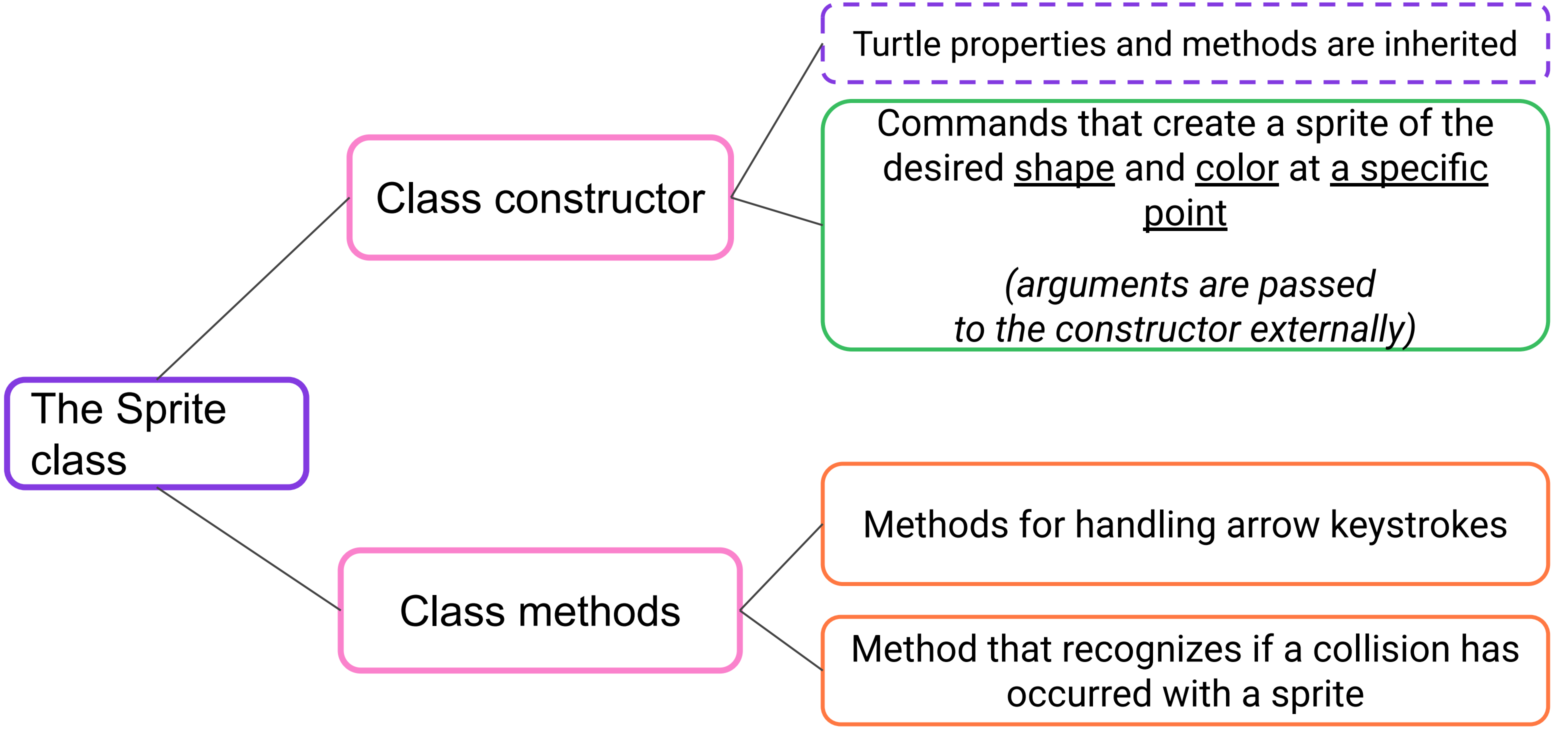


Instances of the new Sprite class (inherited from Turtle).

Brain storming

Designing the Sprite Class

What should the Sprite class be?



The Sprite class

Class constructor

Turtle properties and methods are inherited

Commands that create a sprite of the desired shape and color at a specific point

(arguments are passed to the constructor externally)

Class methods

Methods for handling arrow keystrokes

Method that recognizes if a collision has occurred with a sprite

The diagram can be supplemented as the game is developed

Brain storming

The Sprite class

Let's implement a part of the class to create a sprite with the properties we need (color, shape, position).

```
class Sprite(Turtle):
    def __init__(self, x, y, step=10, shape='circle', color='black'):
        super().__init__()
        self.penup()
        self.speed(0)
        self.goto(x, y)
        self.color(color)
        self.shape(shape)
        self.step = step


player = Sprite(0, -100, 10, 'circle', 'orange')
```

The Sprite class — derived Turtle

Brain storming

The Sprite class

Let's implement a part of the class to create a sprite with the properties we need (color, shape, position).

Hints can be assigned to the parameters

```
class Sprite(Turtle):
    def __init__(self, x, y, step=10, shape='circle', color='black'):
        super().__init__()
        self.penup()
        self.speed(0)
        self.goto(x, y)
        self.color(color)
        self.shape(shape)
        self.step = step

player = Sprite(0, -100, 10, 'circle', 'orange')
```

Sprites can vary in shape, color, starting position, and movement speed.

Brain storming

The Sprite class

Let's implement a part of the class to create a sprite with the properties we need (color, shape, position).

```
class Sprite(Turtle):

    def __init__(self, x, y, step=10, shape='circle', color='black'):
        super().__init__()
        self.penup()
        self.speed(0)
        self.goto(x, y)
        self.color(color)
        self.shape(shape)
        self.step = step


player = Sprite(0, -100, 10, 'circle', 'orange')
```

We inherit all the properties and methods from the superclass.

Brain storming

The Sprite class

Let's implement a part of the class to create a sprite with the properties we need (color, shape, position).

```
class Sprite(Turtle):
    def __init__(self, x, y, step=10, shape='circle', color='black'):
        super().__init__()
        self.penup()
        self.speed(0)
        self.goto(x, y)
        self.color(color)
        self.shape(shape)
        self.step = step
```

We'll create a sprite with the desired properties at a specific point.

```
player = Sprite(0, -100, 10, 'circle', 'orange')
```

Brain storming

Controlling the sprite player

We know how to control the turtle object from the keyboard.

The Sprite object can be controlled the same way because it inherits all the properties and methods from Turtle!

The main program code:

```
scr = player.getscreen()
scr.listen()
```

Create a Screen object (the one that the player sprite is on).

```
scr.onkey(player.move_up, 'Up')
scr.onkey(player.move_left, 'Left')
scr.onkey(player.move_right, 'Right')
scr.onkey(player.move_down, 'Down')
```

We'll handle keystrokes with the corresponding methods of the Sprite class
(add them to the class description).

Brain storming

Controlling the sprite player

We know how to control the turtle object from the keyboard.

The Sprite object can be controlled the same way because it inherits all the properties and methods from Turtle!

The main program code:

```
scr = player.getscreen()
scr.listen()


scr.onkey(player.move_up, 'Up')
scr.onkey(player.move_left, 'Left')
scr.onkey(player.move_right, 'Right')
scr.onkey(player.move_down, 'Down')
```

Please note!

To make the scr object responsive to keystrokes, click on the screen.

After that, the keystrokes will also start "working."

Brain storming

Controlling the sprite player

The Sprite class

```
def move_up(self):
      self.goto(self.xcor(), self.ycor() + self.step)
```

…the sprite should move one step (specified at creation) in the desired direction.

The main program code:

```
scr = player.getscreen()
scr.listen()


scr.onkey(player.move_up, 'Up')
scr.onkey(player.move_left, 'Left')
scr.onkey(player.move_right, 'Right')
scr.onkey(player.move_down, 'Down')
```

When the up arrow key is pressed…

Brain storming

Combining code snippets:

```
class Sprite(Turtle):
    def __init__(self, x, y, step=10, shape='circle', color='black'):
```

Class constructor

```
    def move_up(self):
```

This and other methods that handle arrow key presses

Creating sprites: player, obstacles, and target ')

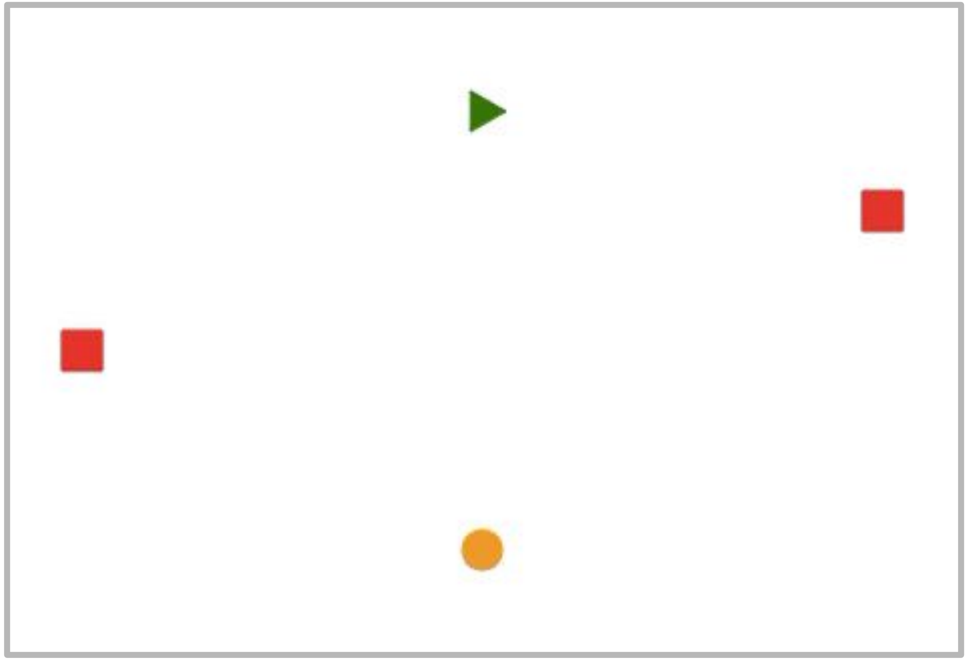
Create a Screen object and "listen" to keyboard events

Subscribing to arrow key events

Brain storming

Tasks:

- → Program the "Hit It!" game with the necessary sprites and the ability to control the player using the keyboard.



Brain storming