Kids Code Convention 2018 Python

by Phil Gilmore

Thanks to our Sponsors!

Mindfire
TECHNOLOGY

UTAH
GEEK EVENTS
training for geeks

NEUMONT
UNIVERSITY

Making Games with Python

To make games in Python at home, you will need:

- Python 3.6
- A text editor (Visual Studio Code)
- A graphics editor (Paint.NET)
- A sound editor (Audacity)

Making Games with Python

For this class, we will use:

- Python 3.6 or newer
  - PyGame package
- Visual Studio Code (the latest, 1.22 or newer)

You won't need anything else for this class. I have created all the graphics and sounds you will need.

Installing GlobalDefense

Download the game contents.

- Navigate your browser to this URL:

  `bit.ly/KCC2018Python`
- Click the DOWNLOAD button, choose Direct Download
- Download the file
- When the file is downloaded, open the file.

Installing GlobalDefense (2)

- Windows:
  - Copy the contents of the downloaded zip file to this folder. You'll need to create it:
    - %userprofile%\globaldefense.

- Mac:
  - Copy the contents of the downloaded zip file to this folder. You'll need to create it:
    - ~/globaldefense

Installing Python, Windows

From the Software Installers\Windows folder, install Python 3.6.5 with these options:

- Custom installation
- [X] Add to System Path
- [X] Install for all users
- Enter custom path: c:\python36
- If prompted, click the button to increase the MAX_PATH.

Installing Python, Mac

From the Software Installers/Mac folder, install Python 3.6.5 with the default options.

Check Python Version

- Windows
  - Click Start
  - Type: *cmd*
  - Type: *python --version*
- Mac
  - Command-space
  - Type: *term*
  - Type: *python --version*

The version should be 3.6.5. If not, fix your system path.

Check PIP Version

- Windows
    - Click Start
    - Type: *CMD*
    - Type: *pip --version*
- Mac
    - Command-space
    - Type: *term*
    - Type: *pip --version*

The version should be 3.x. If not, fix your system path.

Check PyGame

- Windows
  - Click Start
  - Type: *cmd*
- Mac
  - Command-space
  - Type: *term*
- Then
  - Type: *python*
  - Type: *import pygame*
  - Type: *pygame.init()*
  - Type: *quit()*

If there is an error, install PyGame.

Installing PyGame

- Windows
  - Click Start
  - Type: *cmd*
- Mac
  - Command-space
  - Type: *term*
- Then
  - Type: *pip install pygame*

Install Visual Studio Code

From the Software Installs\ directory, run the VS Code installer for your operating system.

Windows:

- VSCodeSetup-x64-1.24.1.exe

Mac:

- VSCode-darwin-stable.zip

Set Up Visual Studio Code (1)

- Open Visual Studio Code.
  - Windows: Click START, type *Visual Studio Code*
  - Mac: Command-Space, type *Visual Studio Code*?
- Select a working folder.
  - Click the Explorer icon in the sidebar (Ctrl-Shift-E)
  - Click the OPEN FOLDER button. We can write our python code here now.
- Open a terminal
  - Press Ctrl-` (the weird one on the ~ key)
  - Click the TERMINAL tab in the tool panel. We can run system commands here now.

Set Up Visual Studio Code (2)

- Install the Python extension:
  - If the sidebar is not open, open it (Ctrl-B)
  - Click the extensions button (the square)
  - If the Microsoft Python extension is not installed, install it:
    - In the extensions window, click on the search box.
    - Type: *python*
    - Click on the Microsoft Python extension. It should be the first one. It should have about 12 million downloads.
    - Click the install button.
    - When installation is finished, click the reload button.

Graphical vs. Textual Dev

Graphical

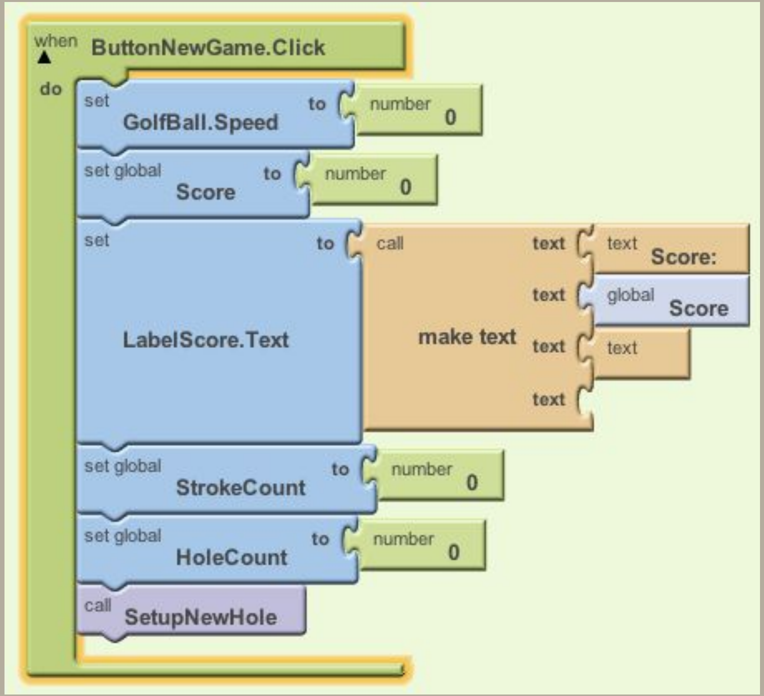
- Scratch
- MIT AppInventor
- Lego Robotics

Textual

- Everything else

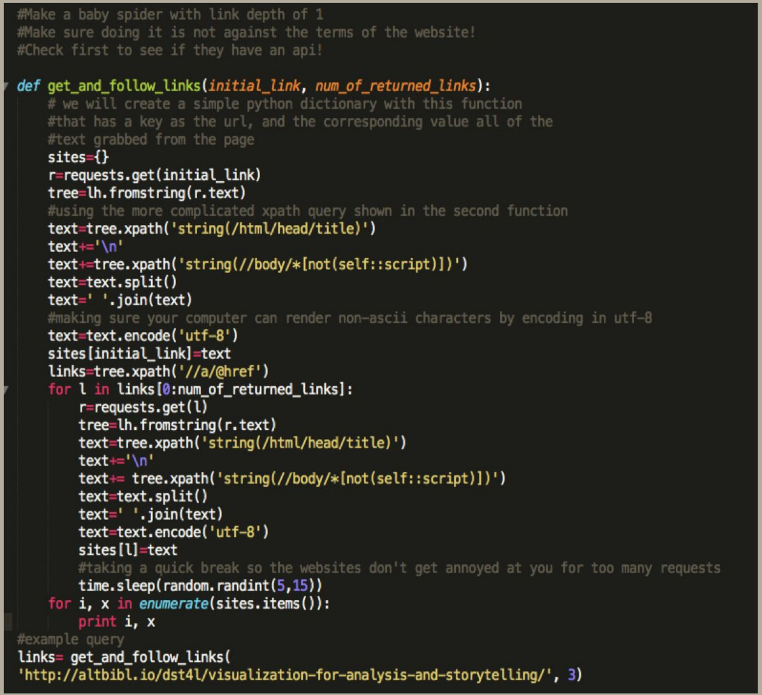
Graphical vs. Textual Dev (cont)

Graphical



when ButtonNewGame.Click
do
set GolfBall.Speed to number 0
set global Score to number 0
set LabelScore.Text to call make text
text text Score:
text global Score
text text
text
set global StrokeCount to number 0
set global HoleCount to number 0
call SetupNewHole

Textual



```
#Make a baby spider with link depth of 1
#Make sure doing it is not against the terms of the website!
#Check first to see if they have an api!

def get_and_follow_links(initial_link, num_of_returned_links):
    # we will create a simple python dictionary with this function
    #that has a key as the url, and the corresponding value all of the
    #text grabbed from the page
    sites={}
    r=requests.get(initial_link)
    tree=lh.fromstring(r.text)
    #using the more complicated xpath query shown in the second function
    text=tree.xpath('string(/html/head/title)')
    text+='\n'
    text+=tree.xpath('string(//body/*[not(self::script)])')
    text=text.split()
    text=' '.join(text)
    #making sure your computer can render non-ascii characters by encoding in utf-8
    text=text.encode('utf-8')
    sites[initial_link]=text
    links=tree.xpath('//a/@href')
    for l in links[0:num_of_returned_links]:
        r=requests.get(l)
        tree=lh.fromstring(r.text)
        text=tree.xpath('string(/html/head/title)')
        text+='\n'
        text+= tree.xpath('string(//body/*[not(self::script)])')
        text=text.split()
        text=' '.join(text)
        text=text.encode('utf-8')
        sites[l]=text
        #taking a quick break so the websites don't get annoyed at you for too many requests
        time.sleep(random.randint(5,15))
    for i, x in enumerate(sites.items()):
        print i, x
#example query
links= get_and_follow_links(
'http://altbibl.io/dst4l/visualization-for-analysis-and-storytelling/', 3)
```

Interpreted Code

Pros

- Runs on many platforms
- Can run things as you go
- Easier languages like Python
- You can still use a compiler if you want to.

Cons

- Players have to have Python installed to play the game
- Your code can be very slow

Interactive Console

- Started from a command / terminal window
  - > Python.exe
- Analyzes and transforms your input as you go
  - Literals
  - Mathematical and boolean expressions
  - Objects
  - Functions
  - Logic and code
- Ctrl-Z or “quit()” to exit the console.

Programming Concepts

Console Input / Output

Display something on the screen:

- print(contentToShow)

Ask the user for input (Python 3):

- user_name = input(“What is your name? “)

Variables

A storage unit with a name. Like a storage shed.

Example:

-
  ```
  >>> a = 5
  >>> a + 2
  7
  ```

Normal Types

All data has a type. We can have whole numbers, decimal numbers or text (strings).

- 42
- 3.141592653589
- “Hello World”

Literals

Values which we type out in our code are called **literals**.

- Example:
  - a = 5
  - b = 7
  - c = a + b
- 5 and 7 are literals because we typed them out in our code. a, b, and c are variables.
- The string "x" has quotes and is literal but *x* is a variable.

More Types: Lists

- A row of storage units.
- Each element has a number, starting with 0
- Units can contain any type, including... lists!
- Like a briefcase. Good for organizing and changing.
- Literal lists are surrounded by [and].

```
list1 = [1]
list1.append(2)
list1.append([3, 4])
list1.extend([5, 6])
del list1[1]
```

```
[1, [3, 4], 5, 6]
```

More Types: Tuples

- Very much like lists, but usually don’t change.
- Useful for grouping related objects, such as name / value pairs or graphical coordinates.
- Like a purse. Good for keeping things together.
- Literal tuples are surrounded by (and).

```
Coordinates = (320, 240)
x, y = coordinates
print(x)
print(y)
```

```
320
240
```

Operators

(+) Addition

(-) Subtraction

(*) Multiplication

(/) Division

(%) Modulus (remainder)

(**) Power (exponent)

(//) Integer division

= Assignment

== Equality comparer

!=, <> Inequality comparer

< Less than

<= Less than or equal to

> Greater than

>= Greater than or equal to

Boolean Logic

An element of a boolean type can have either of two values:

- True
- False

Boolean AND Logic

Truth table
(AND == “if both operand1 AND operand2 are true then result is true”)

True AND True == True

True AND False == False

False AND False == False

Boolean OR Logic

Truth table

(OR = “If either operand1 OR operand2 is True then result is True”)

True OR True == True

True OR False == True

False OR False == False

Code Blocks

Code blocks have a BEGINNING and an ENDING. All lines between are a block.

```
C#
{
    CodeBlock
}
```

```
SQL
BEGIN
    CodeBlock
END
```

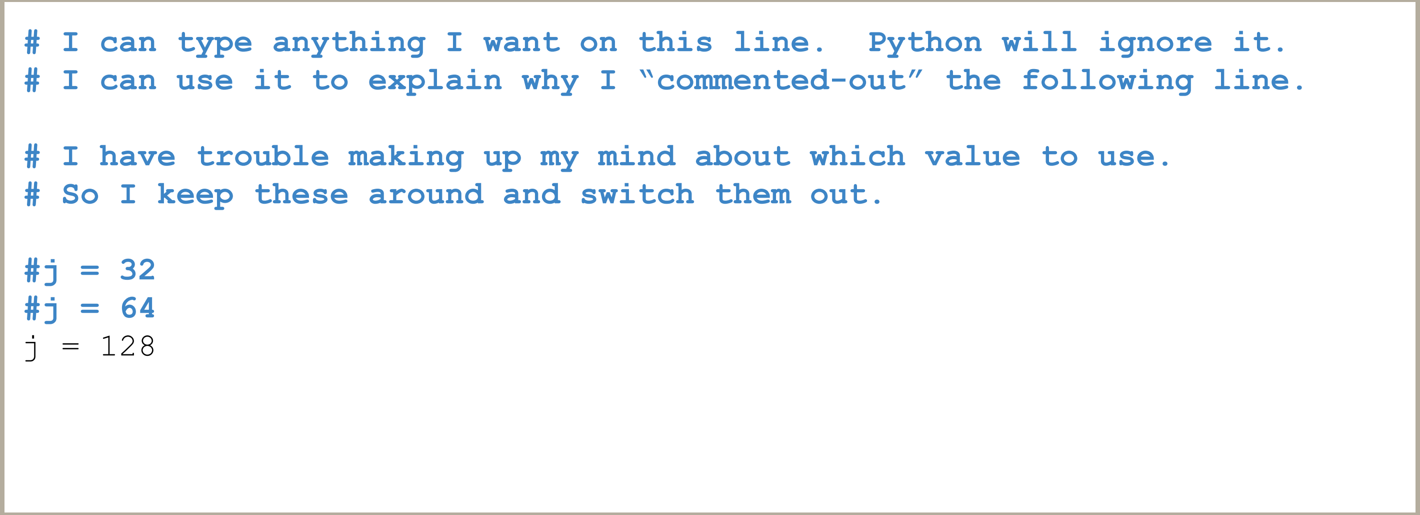
```
Java {
    CodeBlock
}
```

```
Python:
    CodeBlock
```

Comments / Remarks

Lines that start with # are ignored.

- Temporarily disable lines of code.
- Add human-readable remarks or explanations.



```
# I can type anything I want on this line.  Python will ignore it.
# I can use it to explain why I “commented-out” the following line.

# I have trouble making up my mind about which value to use.
# So I keep these around and switch them out.

#j = 32
#j = 64
j = 128
```

Functions

We take a list of instructions and put them in a box with a name. We can run the instructions over and over just by using the name.

Functions

How do we brush our teeth?

```
Go to bathroom
Turn on light
Fetch brush
Wet brush
Fetch toothpaste
Open toothpaste
Apply toothpaste to brush
Scrub tooth 1, 2, 3, 4, … 32
Spit (ew)
Rinse brush
Rinse mouth
Spit (ahh!)
Rinse sink (no minty chunks)
Return brush and toothpaste to cupboard
Turn off light
```

Functions

Does Mom tell you do do each of those things? Of course not. She just says “brush” and you know the list of instructions.

```
brush_your_teeth()
```

Functions

Sometimes we have questions about how to do the function. The caller can give us that information as parameters between the (and the).

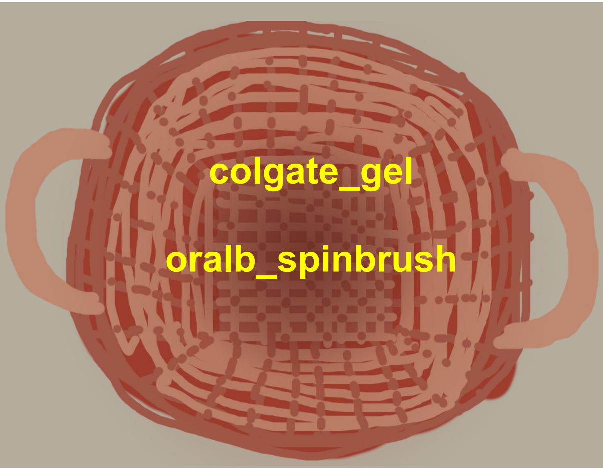
```
brush_your_teeth(colgate_gel, oralb_spinbrush)
```

Think of the (and) as handles of a basket which the caller gives you. Whatever's in it, you can use.

Functions

Think of the (and) as handles of a basket which the caller gives you. Whatever's in it, you can use while executing the instructions.

```
brush_your_teeth(colgate_gel, oralb_spinbrush)
```



Functions

Here’s how the function looks in Python:

```
def brush_teeth(toothpaste_to_use, brush_to_use):
    Go to bathroom
    Turn on light
    Fetch brush_to_use
    Wet brush_to_use
    Fetch toothpaste_to_use
    Open toothpaste_to_use
    Apply toothpaste_to_use to brush_to_use
    Scrub tooth 1, 2, 3, 4, … 32
    Spit (ew)
    Rinse brush_to_use
    Rinse mouth
    Spit (ahh!)
    Rinse sink (no minty chunks)
    Put brush_to_use and toothpaste_to_use in cupboard
    Turn off light
```

Functions

Functions can also return a value:

```
def add_numbers(addend1, addend2):
     result = addend1 + addend2
     return result
print(add_numbers(5, 2))
```

```
7
```

Python Code Blocks

- Starts with a colon (:) on the previous line.
- Indentation determines which lines are in a code block.
- Indentation must be identical to other lines in the block.
- Indentation must be greater than the code outside the block.

```
#Outside the block
dostuff()
def f():
    #Inside the block
    dostuff2()
    dostuff3()
#Outside the block
dostuff4()
```

Classes

A class:

- Has a Name.
- Is another box, like a function but bigger.
- Can contain functions and variables.
- Can contain smaller classes. This is called inheritance.
- Is a blueprint for building objects in its own image.
- Has special rules
- PyGame uses them to make sprites.

Sprites

- It’s a graphical picture.
- Has transparent background instead of a solid square behind them.
- Can be drawn anywhere on the screen
- Can collide with (touch) other sprites.
- Knows how to move itself
- Knows how to animate itself

Consuming Classes

```
h = Hello()
h.hello_world()
```

Programming Tasks

Conditional Execution

To determine which code to execute based on a condition, use the “if” statement.

```
if True:
     then_this_will_execute()
```

```
if True:
     then_this_will_execute
else:
     this_will_execute_instead()
```

```
if condition1:
     condition1_is_true()
elif condition2:
     condition2_is_true()
else:
     neither_are_true()
```

Looping (For)

Apply a task to every item in a collection using FOR.

```
name_list = ["Phillip", "Nathan", "Daniel"]

for name in name_list:
     print(name)
```

```
Phillip
Nathan
Daniel
```

Looping (Classic For)

Apply a task to each number in a range using FOR.

```
for index in range(3):
     print(index)
```

```
0
1
2
```

Looping Until Finished (WHILE)

Loop as many times as necessary. In other words, loop forever until something makes us stop. Do this with a WHILE loop.

```
j = 1
while (j < 303):
     if (j == 77):
          print("Found: " + str(j))
          break
     j = j + 1

if j >= 303:
     print("None found.")
```

```
77
```

Modules

- Write code that performs a task.
- Save it to a .py file.
- Reuse that code in other .py files.

```
# This code is saved in a file called:
# philmath.py

pi = 3.141592653589

def deg_to_rad(degrees):
     return degrees / 180 * pi
```

```
import philmath
print(philmath.pi)
```

```
from philmath import deg_to_rad
print(deg_to_rad(180))
```

Game Concepts

Game Structure

Structure of a game:

- One-time setup
  - Create sprites, graphics, sounds
- Endless loop
  - Look for game events (keyboard & mouse input)
  - Move & change graphics (sprites) in response to input
  - Draw graphics
  - Play sounds
  - Update the screen with new graphics
  - Pause to regulate game speed
  - Repeat

Graphics

Uses a cartesian plane.

Only the +x,+y quadrant is shown on the screen.

The origin at 0,0 is the upper-left of the screen.

+x moves right

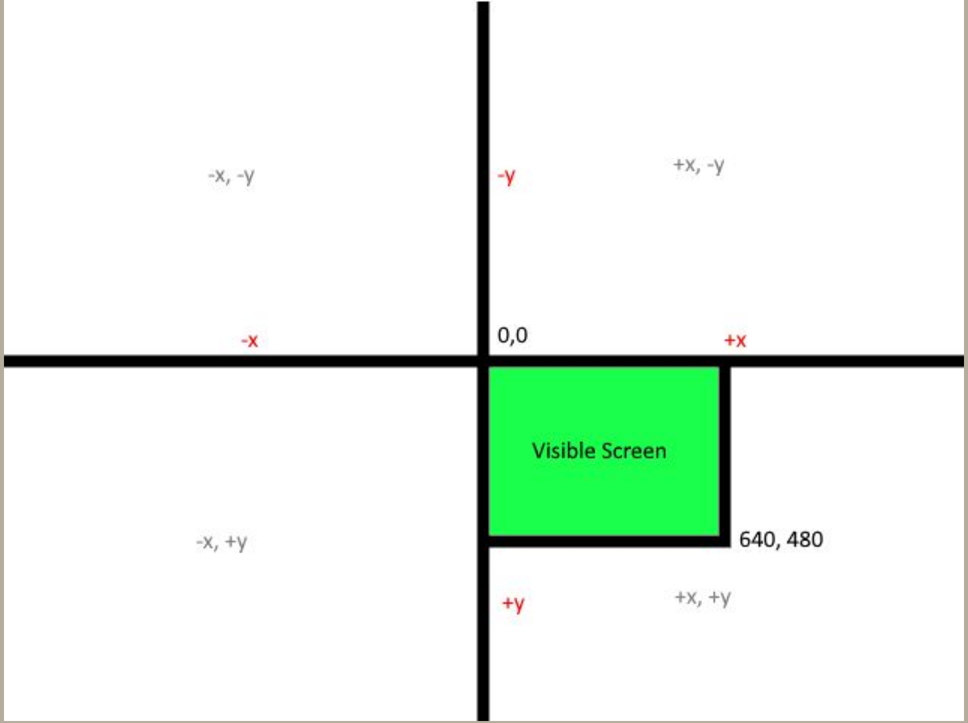
+y moves down

-x moves left

-y moves up

Graphics

Cartesian Plane



Surfaces

Rectangular graphics object.

Can be copied onto the screen at any location.

When copying, we call it a BLIT.

This includes:

- Text
- Sprites

Rectangles (Rect)

Rect is an object. Every PyGame surface has one.

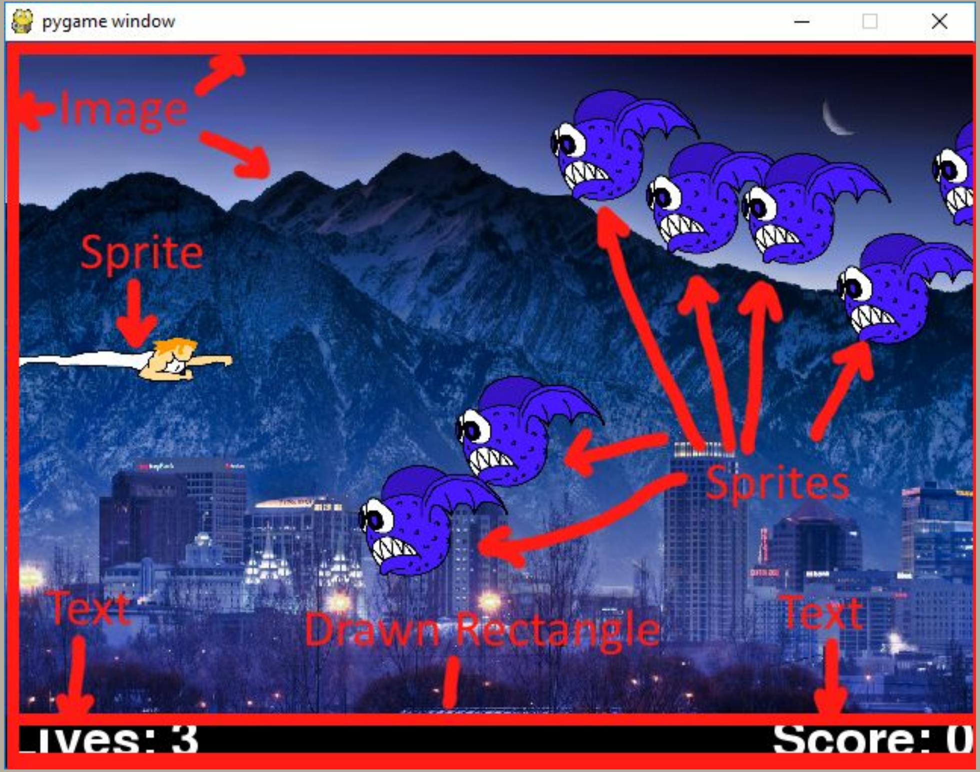
It includes the size and location of the surface.

Obtained by calling *surface.get_rect()*

Properties:

- top, left, bottom, right
- width, height, centerx, centery

Element Examples



Collisions

To figure out if two sprites are touching, we compare their rectangles. PyGame can do this for us.