Python Course ~ Zero to Knowing

Learn Object Oriented Programming through project based learning

Some Rules for Python to follow:

Beginning Rule	The first command of the code is written at the top left, like a book
Order Rule	Commands are executed in the order they are written
Style Rules	Replacing lowercase letters with uppercase is not allowed. A space or comma could break the code

Print() - is a pre-programmed function ready-to-use

Syntax	Output
print(“Hello, world!”)	Hello, world!
print(“Hello”, “world!”)	Hello world!
print(“I am: 55”)	I am: 55
print(78, 6)	78 6

- Use Double Quotes for text -> “Hello”
- A comma (,) will create an automatic space in the output

Print() - is a pre-programmed function ready-to-use

Syntax	Value	Output
print(3 * 9)	Multiplication	27
print(23 + 4)	Addition	27
print(36 - 9)	Subtraction	27
print(54 / 2)	Division	27.0
print(“Grand Total:”, 25*4)		Grand Total: 100

Print() - is a pre-programmed function ready-to-use

Syntax	Value	Output
print(21 % 2)	Remainder	1
print(22 // 3)	Quotient -> Rounded down	7
print(4**4)	Raising to a power	64

PEMDAS -> Please Excuse My Dear Aunt Sally

Parentheses, Exponents, Multiplication, Division, Addition, Subtraction

Intro to Variables

Working with Variables in Python

Variables

- A **Variable** is a name that holds a value
- It’s a data element that has its own name
- We use Variables to work with changing data

Defining Variables:

Using Variables we must do the following: -> A variable is equal to a value

- Create a variable by giving it a name (Lowercase, No numbers, No special characters)
- Set the value of the variable

Example:

Variable name

```
years = 2
months = 6

print("I've had this job for", years, "years and", months , "months")
```

value

Assignment operator:

To set a variable's value, you must to use the assignment operator which is =

Variable

Operator

Value

```
years = 2

print(years)
```

Output in console
2

Assignment operator:

*The equal (=) operator can <u>change the value</u> of an existing variable

Remember -> Python reads Top to Bottom

```
years = 2

years = 5

print(years)
```

Changing the initial value

Output in console
5

Naming of Variables:

- You can use letters, digits, and underscores
- Every variable's name must begin with a letter
- A variable's name can't be a preexisting command and other special syntax, such as, print or input

Naming of Variables:

Good naming conventions:

Bad Names	Reason
p = 25 num = 8.46	Too Generic, be specific
number_of_airports_in_america = 861	Too long

Effective names	Reason
lastName = 'Gates' item_cost = 99 amount_sold = 222	The names explain the purpose Other programmers will be able to read your code

Types of Data:

Variables can hold different types of data. You now know three:

- **integer** numbers -> A whole number
- **decimals** -> Also called float
- **string** data -> Text

Numbers		Strings
563	Integer	“James Woods”
27.6	Decimal (float)	“123”
3 * 22	Integer	“10 - 25 - 2001”
5 * 7.3	Decimal (float)	“Good Input”

Numbers as a value:

Example: The following code calculates a passengers flight fare

```
base_fare = 155
num_of_bags = 2
base_fare = 205
total = base_fare * (num_of_bags * 0.75)
print(total)
```

Changing Variable Value

Output in console
307.50

Entering Data

How to gather information from the user with data input

The input() function:

input() allows for a <u>user to enter data</u> -> Python asks the user a question and can use their answer

```
lastName = input("Enter the owner's last name:")
city = input("Enter the city:")
print("The owner in", city, " is", lastName)
```

Smith

Seattle

The Output
"The owner in Seattle is Smith"

The input() function:

input() allows for a <u>user to enter data</u>

```
password = input("Password:")
```

Variable that holds input value

function

A hint (Placeholder Text)

The input() function:

Errors because of different data types

```
extra = input('Price of extra add-ons:')
total = 1000 + extra
print('Total price:', total)
```

The value of input is a string, <u>not a number</u>

Computers can’t add up integer and string values

We need to convert the string values into integer values before any math

Convert Functions -> int() and str()

int() and str() functions are used to switch between numbers and strings

```
extra = input("Price of add-ons:")
extra = int(extra)
total = 1000 + extra
print("Total price:", total)
```

Converts the input (string) to an integer

1. Extra input is converted from a string to an integer
2. We can now add **1000** and **extra** together as normal
3. **str()** converts an integer to a string (opposite)

Nesting Functions

Creating shorter and cleaner code with Nesting

Putting functions inside of each other

How do we nest:

Nesting -> The act of putting functions inside each other to create shorter code

Understand the first two things:

1. Know which functions can be nested
2. How to form them correctly

```
height = int(input("Enter height: "))
```

The Two return values:

Functions that work with data types

Returns no “real” value

Return a useable value

```
print(“July 2000”)
```

print() doesn’t return a value the computer can use. Only a readable value

```
time = input(“Time: ”)
```

→ String

```
cost = cost_1 + cost_2
```

→ Number

```
length = len(string)
```

→ Number

Format our nested functions:

```
height = input("Enter your height: ")
```

1. This results in a **string**

```
height = int(height)
```

2. The result is an **integer**

```
height = int(input("Enter your height: "))
```

The code performs the same actions

Examples of Nesting:

Example: Finding the cost based on number of days

```
cost = input("Cost: ")
cost = int(cost)
length = input("Number of days: ")
length = int(length)
total = cost * length
print(total)
```

```
cost = int(input("Cost: "))
length = int(input("Number of days: "))
total = cost * length
print(total)
```

Better code because we used nesting

More examples of Nesting:

Example: Finding the sum of the input length based on number of letters

```
word = input("Enter a random word: ")
color = input("Enter a random color: ")
length_1 = len(word)
length_2 = len(color)
total = length_1 + length_2
print("Total letters:", total)
```

```
length_1 = len(input("Enter a random word: "))
length_2 = len(input("Enter a random color: "))
total = length_1 + length_2 + length_3
print("Total letters:", total)
```

Readable Code with comments:

A **comment** is basically "Notes" in your code

```
#a short note/comment
```

Single-line comment.

```
'''A long note/comment explaining
the code'''
```

Multi-line comment.

Example:

```
cost = int(input("Enter the cost for goods: ")))
#sale- 20% off the goods
print("Savings Earned:", cost * 0.2)
```

VS code sees # and skips it

Conditional Statements

<u>If something is True, do this. Else do this instead</u>

Computers only understand two things, 0 and 1

1 = True

2 = False

Understanding Conditions:

A condition is either True or False

This is known as a Boolean Value (True or False)
Or in other words a "Logical Expression"

If something is True**:**
    Do this

Else if that is not true**:**
    Do this instead

Simple expressions:

Example: If available_tickets is **greater than** sold_tickets then that is **True**

```
available_tickets = int(input("Number of available tickets:"))
sold_tickets = int(input("Tickets sold: "))
enough = available_tickets > sold_tickets
print(enough)
```

100

75

Logical expression

Output in console
True

Using operators to compare values:

Logic					
>	<	<=	>=	==	!=
Greater than	Less than	Less than or equal	Greater than or equal	**Equal** To	**Not Equal** To

Numbers					
*	/	+	-	%	//
Multiplication	Division	Addition	Subtraction	Remainder	Division (rounded down)

Additional Logical Operators:

And / Or logical operators

Name of Operator	When to use
and	When two conditions need to be **True** at the **same time**
or	When only **one condition** needs to be **True**

```
availiable_tickets = int(input("Number of available tickets:"))
buy_more = available_tickets < 75 or available_tickets > 500
print("Ticket Warning:", buy_more)
```

The basics of Conditional Statements:

A condition **only** runs if the expression is True

if This is True (Expression) :

Code after colon runs "if True"

Run this code

1 Tab

After a colon, it will **automatically** tab your next line of code when you click "Enter"

if This is True (Expression) :

Run this code

else :

Run this code instead

1 Tab

Think "Nesting"

The basics of Conditional Statements:

Example: The user enters a type of music. If the users enters **“rock”**, python replies with “AC/DC”. **Anything else entered**, python replies with “Eminem”

```
genre = input(“Music Genre: ”)
if genre == “rock”:
    print(“AC/DC”)
else:
    print(“Eminem”)
```

rock

Output in console
AC/DC

*** When checking **“if something is True”**, we always use **==** 2 equal signs

The basics of Conditional Statements:

!= is to check for something that is **NOT EQUAL**

```
password = input("Enter your password: ")
if password != "7753":
    print("Incorrect, Password!")
else:
    print("Welcome back to your account!")
```

9983

If input **is not equal** to "7753"

Output in console
Incorrect, Password!

Nesting Conditions

```
remember =  int(input(“Enter number here: “))
```

Special string methods (functions)

How to convert a string input to lower or uppercase?

.lower() → Converts the input to all lowercase

.upper() → Converts the input to all uppercase

.capitalize() → Will capitalize the first letter of a string (first word)

These are functions known as “methods” -> notice the **.** before the function

*A <u>method</u> **must be linked** to a <u>variable</u> to work

Nesting Conditions:

```
if This is True (Expression) :
    if This is True (Expression) :
        Run this code
    else :
        Run this code instead
else:
    Run this code instead
```

1 Tab

If this is **True**, then

Check to if this is **also True**

If **that was False**, Run this code

If **that was False**, Run this code

Nesting -> Putting code inside of each other

Nesting Conditions:

Example:
If the **user enters "beach"**, then python replies "Visit Thailand". If the user **enters anything different** from "beach", python asks the user "warm or cold?". If the **user enters "cold"**, python replies with "A Japan ski holiday" **anything else**, python replies with "A mountain holiday to the Rockies"

```
destination = input("Enter a travel destination: ").lower()
```

City trip

```
if destination == "beach":
    print("Visit Thailand")
else:
    dream = input("Would you prefer warm or cold weather? ")
```

Cold

```
    if dream == "cold":
        print("A Japan ski holiday")
    else:
        print("A mountain holiday to the Rockies")
```

Output in console
???

Nesting Conditions:

Example:
If the **user enters "beach"**, then python replies "Visit Thailand". If the user **enters anything different** from "beach", python asks the user "warm or cold?". If the **user enters "cold"**, python replies with "A Japan ski holiday" **anything else**, python replies with "A mountain holiday to the Rockies"

```
destination = input("Enter a travel destination: ").lower()
```

City trip

```
if destination == "beach":
    print("Visit Thailand")
else:
    dream = input("Would you prefer warm or cold weather? ")
    if dream == "cold":
        print("A Japan ski holiday")
    else:
        print("A mountain holiday to the Rockies")
```

Cold

I entered "Cold"... Why?

Output in console
A mountain holiday in the Rockies

Nesting Conditions:

Example:
If the **user enters "beach"**, then python replies "Visit Thailand". If the user **enters anything different** from "beach", python asks the user "warm or cold?". If the **user enters "cold"**, python replies with "A Japan ski holiday" **anything else**, python replies with "A mountain holiday to the Rockies"

```
destination = input("Enter a travel destination: ").lower()
```

City trip

```
if destination == "beach":
    print("Visit Thailand")
else:
    dream = input("Would you prefer warm or cold weather? ")
    if dream == "cold":
        print("A Japan ski holiday")
    else:
        print("A mountain holiday to the Rockies")
```

Cold

We forgot to make our input **lowercase**.

Output in console
A mountain holiday to the Rockies

Nesting Conditions:

Example:
If the **user enters "beach"**, then python replies "Visit Thailand". If the user **enters anything different** from "beach", python asks the user "warm or cold?". If the **user enters "cold"**, python replies with "A Japan ski holiday" **anything else**, python replies with "A mountain holiday to the Rockies"

```
destination = input("Enter a travel destination: ").lower()
```

City trip

```
if destination == "beach":
    print("Visit Thailand")
else:
    dream = input("Would you prefer warm or cold weather? ").lower()
    if dream == "cold":
        print("A Japan ski holiday")
    else:
        print("A mountain holiday to the Rockies")
```

Cold

Output in console
A Japan ski holiday

Multiple Conditional Statements:

```
if  This is True (Expression)  :
        Run this code

elif  else if this is True (Expression 2)  :
        Run this code

else :
        If neither of the above are
            True, Run this code
```

elif means “Also if” or “else if”

elif can **only** come after “if” and before “else”

Multiple Conditional Statements with Logic:

```
price = int(input("Price: "))
days_spent = int(input("Number of days spent: "))
if days_spent >= 3 and days_spent <= 5:
    price = price * 0.80
    print("Your discounted price:", price)
elif days_spent >= 6 and days_spent <= 9:
    price = price * 0.75
    print("Your discounted price:", price)
else:
    print("No Discount, your total is:", price)
```

Multiple Conditional Statements with Logic:

```
price = int(input("Price: "))
days_spent = int(input("Number of days spent: "))
if days_spent >= 3 and days_spent <= 5:
    price = price * 0.80
    print("Your discounted price:", price)
elif days_spent >= 6 and days_spent <= 9:
    price = price * 0.75
    print("Your discounted price:", price)
else:
    print("No Discount, your total is:", price)
```

Checks if input() was from **3 thru 5**

If yes, applies a 20% discount

Checks if input() was from **6 thru 9**

If yes, applies a 25% discount

Any other input this is returned

Loops in Python

Repeating code made easy!

While Loop

While something is True, repeat

The While Loop:

- **While** something is **True**, **continue** to run the code
- **Quits** running when it becomes **False**

while <u>This is true (Expression)</u> :

Code after colon runs "while True"

Keep running this code

First thing to run when the expression becomes False

1 Tab

Best loop for games or programs that **require repeating**

The While Loop:

Example: if the password entered **is not equal to** “password123” the code will **repeat** asking the user to “try again”. Once the **user enters “password123”** the loop will **break** and print -> “Welcome to your account”

```
password = input(“Enter your password: ”)
while password != “password123”:
    print(“You entered a wrong password”)
    password = input(“Try again: ”)
print(“Welcome to your account!”)
```

rambo321!

Output in console
You entered a wrong password Try again:

The While Loop:

Example: if the password entered **is not equal to** “password123” the code will **repeat** asking the user to “try again”. Once the **user enters “password123”** the loop will **break** and print -> “Welcome to your account”

```
password = input(“Enter your password: ”)
while password != “password123”:
    print(“You entered a wrong password”)
    password = input(“Try again: ”)
print(“Welcome to your account!”)
```

~~rambo321!~~ → password123

Output in console
You entered a wrong password Try again: password123 Welcome to your account!

The While Loop:

Example: The user will **enter the cost** of each item. While **(as long as)** the user **does not** enter "0", the code will **continue** to run

Once the user **enters "0"**, the code will **stop** and take the **total variable** and multiply it by **50%**

```
cost = int(input("Enter the cost of an item (0 to end): "))

total = 0

while cost != 0:

    total += cost

    cost = int(input("Enter the cost of an item (0 to end): "))

print("Grand Total: ", total)

total = total * 0.50

print("Total price with discount:", total)
```

Output in console
???

The While Loop:

```
cost = int(input("Enter the cost of an item (0 to end): "))
total = 0
while cost != 0:
    total += cost
    cost = int(input("Enter the cost of an item (0 to end): "))
print("Grand Total: ", total)
total = total * 0.50
print("Total price with discount:", total)
```

A counter variable, used for counting

Literally means **total + cost** is now the new **total**

total is now equal to **total + cost**

Takes final total value and multiplies by 50%

Output in console
23

Example: 5 10 4 7 20 0

total = 46

The While Loop with a counter:

Using a counter to track

```
test_answer = input("Enter a, b, c or d: ")
try = 1
while test_answer !=  'c':
    try += 1
    print("Wrong answer!")
    test_answer = input("Try again: ")
print("It took you", try, "tries to get the answer")
```

Counter variable

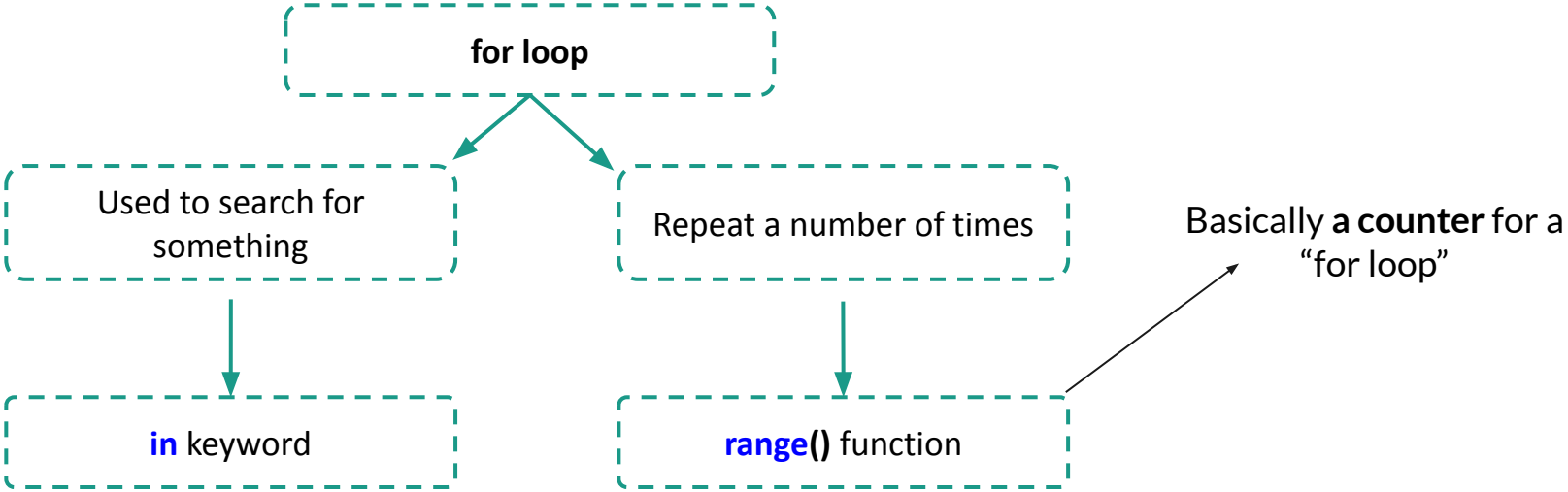
Adding 1 **every time** the answer is wrong

Prints off number if tries it took the user

For Loop

For every element in this variable, do something with it

What is a for loop?



A loop iterating over every item in a data type (strings, lists, dictionaries)

The for loop structure:

A **local variable** we create to use in the **for loop**

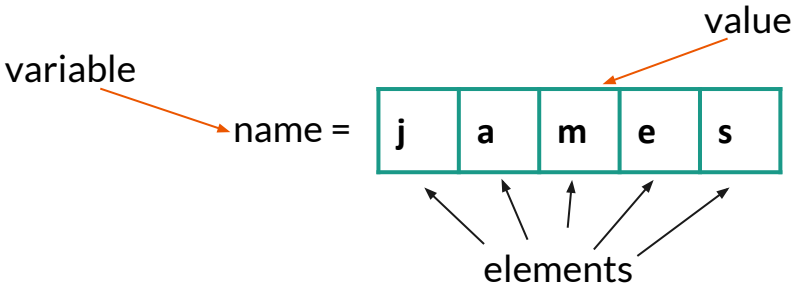
for element in sequence:

What are you searching through?

- String
- Variable
- *List
- *Dictionary

Run this code

Once loop ends, run this code



Literal Translation

for element in sequence:

Run this code

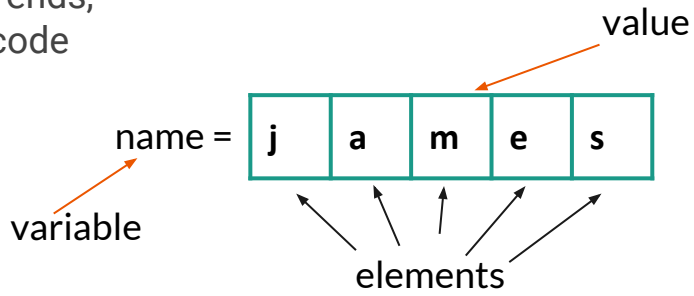
Once loop ends,
run this code

for Every element in My variable:

Run this code

Once loop ends,
run this code

Python Lingo



The for loop structure:

```
username = input("Enter username: ")
invalid = "!@#$%^&*()-_+="
for letter in username :
    if letter in invalid:
        print("This character is not allowed:", letter)
```

j	o	s	h	#	1	2	3

Python with go through every element in my input

This triggers my print() function

The for loop structure:

```
username = input("Enter username: ")
invalid = "!@#$%^&*()-_+="
for letter in username :
    if letter in invalid:
        print("This character is not allowed:", letter)
```

For **every letter** in my username

Check **if** the **current letter** is inside **"invalid"**

If **it is**, then **print** this

j	o	s	h	#	1	2	3

Python with go through every element in my input

This triggers my print() function

The for loop with range() function:

for element in range(range (number)):

Run this code range number of times

Once loop ends, run this code

range(number) – creates a list of numbers – 1,2,3,4 …

range(num1, num2) – Creates a list of numbers starting at num1 – num1 + 1, num1 + 2, num1 + 3…

The for loop with range() function:

```
for i in range(5):
    number = input("Enter a number: ")
    print("Number added:", number)
print("All 5 numbers added!")
```

Asks the user to **input** a number **5 times**

```
passengers = int(input("How many passengers?"))
for i in range(passengers):
    lastName = input("Enter last name: ")
    print("Hello,", lastName)
print("Passenger Manifest Updated!")
```

5

range is now 5

Prints hello 5 times

"i" is a popular element name for representing elements in a list

"i" actually means 'index' which is like saying position

Create an AI trip planner A.K.A -> KayakGPT

Use everything we've learned so far!

Tips for trip planner! (KayakGPT)

Structure:

- Input statement to start the system
- Main While Loop (Repeat until an event)
- Conditional Statements (if/elif/else)
- print()
- Try adding the for loop
- 5 or more main actions
- Nesting is key!

*Use everything we have learned so far!

*Minimum of 50 lines of code

```
enter = input("Enter 1 - start or 2 - stop ")

while enter != "2":
    destination = input("Do you have a place in mind? ").lower()
    if destination == "yes":
        transport = input("Plane/Train or Car?").lower()
    elif action == "no":
        trip_type = input("Beach/City or Adventure?").lower()
        if trip_type == "beach":
            print("Head to Hawaii")
            beach = input("Type of beach (popular/remote): ").lower()
            if beach == "popular":
                print("Great, check out Waikiki beach")
    elif beach == "remote":
        #Code more here!
```

Creating your own Functions

We need to define our own functions, not just use the ready-made python functions

Functions we have used so far:

What do these do?

print()

int()

str()

range()

len()

input()

Functions we have used so far:

What do these do?

print()

Prints off data we can read

int()

Converts a string to an integer

str()

Converts an integer to a string

range()

Repeat a number of times

len()

Returns the length of a string/type

input()

Allows a user to input data

What is a Function?

- A piece of **reusable code** with a **unique name**
- Can be **called/used** in other parts of our code

```
def test_function( parameter1 , parameter2 ) :
    Run this code when function is used
    Run this code when function is used

test_function( argument1 , argument2 )
```

Defining the function
(what will it do?)

Calling/using the function

What is a Function?

We give our function a unique name

- Create **local variables** called **parameters**
- These will be used when we call the function

```
def test_function( parameter1 , parameter2 ):
    Run this code when function is used

    Run this code when function is used

test_function( argument1 , argument2 )
```

We pass in the data that will be **used by the parameters** (arguments)

What is a Function?

```
def person_info( name , age, nationality ):
    print("Welcome:" , name)
    print("You are:", age)
    print("You are from:", nationality)


number = int(input("Amount: "))
for i in range(number):
    name = input("Enter first name: ")
    age = input("Enter your age: ")
    nationality = input("Enter your nationality: ")
    person_info( name , age , nationality )
```

We **def**ine (create) a function called person_info

This code **will not** run until we **call** (use) the function name

Here we **call the function** and **pass it 3** arguments to be used

Must have 3 **arguments** because we made 3 **parameters**

What is a Function?

```
def person_info( name , age, nationality ):
    print("Welcome:" , name)
    print("You are:", age)
    print("You are from:", nationality)


number = int(input("Amount: "))
for i in range(number):
    name = input("Enter first name: ")
    age = input("Enter your age: ")
    nationality = input("Enter your nationality: ")
    person_info( name , age , nationality )
```

That data is passed from the function call to where we defined the function to be used

Here we **call the function** and **pass it 3** arguments to be used

What is a Function?

```
def person_info( name , age, nationality ):
    print("Welcome:" , name)
    print("You are:", age)
    print("You are:", nationality)
```

Output in console
Welcome: Josh You are: 25 You are: American

```
number = int(input("Amount: "))
for i in range(number):
    name = input("Enter first name: ")
    age = input("Enter your age: ")
    nationality = input("Enter your nationality: ")
    person_info( name , age , nationality )
```

Josh

25

American

Return -> operator to get back a value

```
def total_points(game_score):
    points = 0
    while game_score != 0:
        if game_score >= 5 and game_score <= 10:
            points += 2
        if game_score > 10 and game_score <= 20:
            points += 3
        game_score = int(input("Enter the game score: "))
    return points

score = int(input("Enter the score: "))
game_points = total_points(score)
print(points)
print(game_points)
```

Local & Global Variables

Local means near you

A Local variable is only accessible inside of something. It can not be used anywhere else

Global means everywhere

A global variable can be used anywhere in your code. Inside and outside of functions/conditions

Return -> operator to get back a value

```
def total_points(game_score):
    points = 0
    while game_score != 0:
        if game_score >= 5 and game_score <= 10:
            points += 2
        if game_score > 10 and game_score <= 20:
            points += 3
        game_score = int(input("Enter the game score: "))
    return points

score = int(input("Enter team name: "))
game_points = total_points(score)
print(points)
print(game_points)
```

Local & Global Variables

Local means near you

A Local variable is only accessible inside of something. It can not be used anywhere else

Global means everywhere

A global variable can be used anywhere in your code. Inside and outside of functions/conditions

ERROR: We are trying to print a Local Variable

Return -> operator to get back a value

```
def total_points(game_score):
    points = 0
    while game_score != 0:
        if game_score >= 5 and game_score <= 10:
            points += 2
        if game_score > 10 and game_score <= 20:
            points += 3
        game_score = int(input("Enter the game score: "))
    return points

score = int(input("Enter team name: "))
game_points = total_points(score)
print(game_points)
```

Local & Global Variables

Local means near you

A Local variable is only accessible inside of something. It can not be used anywhere else

Global means everywhere

A global variable can be used anywhere in your code. Inside and outside of functions/conditions

Return -> operator to get back a value

```
def good_deal(cost):
  if cost >= 50 and cost < 150:
      response = “This is a good deal!”
  elif cost >= 150:
      response = “Overpriced!”
  else:
      response = “Cheap, Buy Now!”
  return response
```

```
store = input(“Enter Store Name: ”)
cost = float(input(“Item Cost: ”))
res = good_deal(cost)
print(store, “-”, res)
if res == “This is a good deal!”:
   print(“Buy before it’s too late!”)
```

float() -> **New function** for working with numbers, it converts a string to a decimal number

Return -> operator to get back a value

parameter

```
def good_deal(cost):
    if cost >= 50 and cost < 150:
        response = "This is a good deal!"
    elif cost >= 150:
        response = "Overpriced!"
    else:
        response = "Cheap, Buy Now!"
    return response
```

We can use the value of response in other parts of our code

```
store = input("Enter Store Name: ")
cost = float(input("Item Cost: "))
res = good_deal(cost)
print(store, "-", res)
if res == "This is a good deal!":
    print("Buy before it's too late!")
```

This will return a useable value

The Value of the res variable is a function. Whatever the response variable returns, we can use as the new value for 'res'

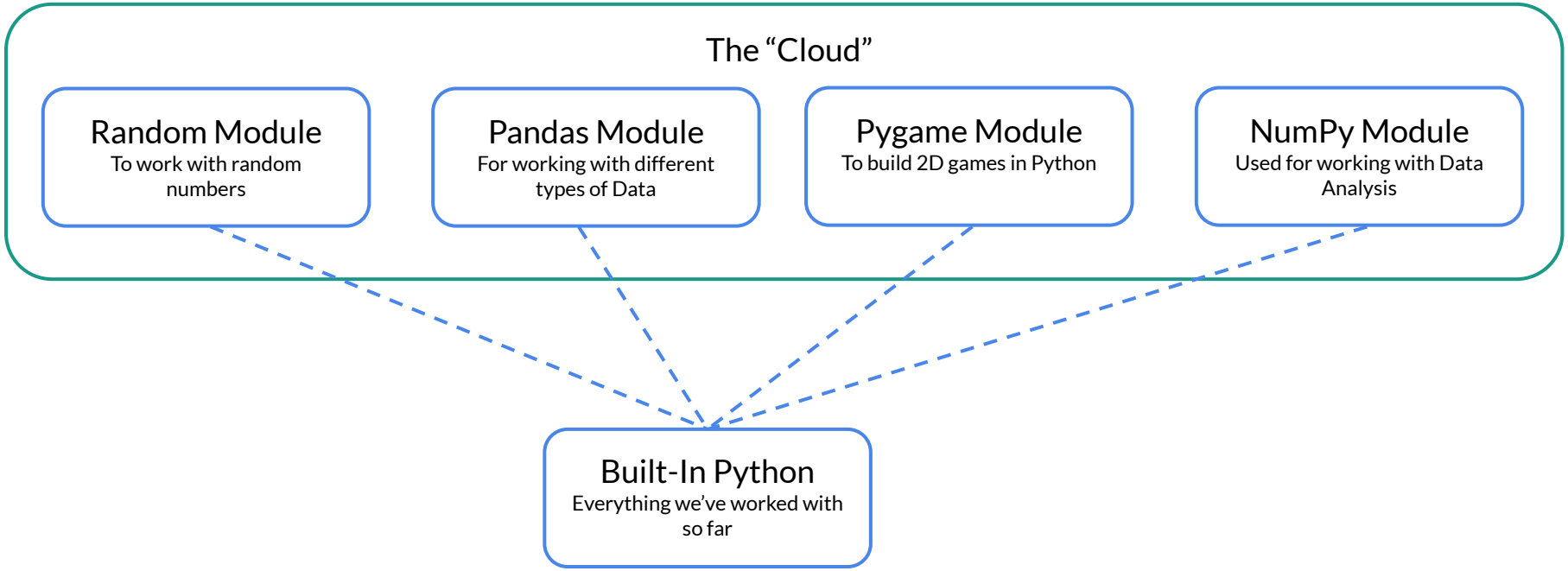
*You **can** call a function within another function!

Working with Modules

A module is extra python code kept in the “cloud”

Modules in Python

Extra code to complete special tasks that we can import into our code to use in our projects with pre-programmed functions



How to import a Module?

Best when learning how to use modules

import random
Connects the entire module, but we still need to state which module it's from

Best when working with larger modules (doesn't over run your system)

from random import shuffle
Imports one specific function (shuffle) from the library random we can use

Most popular way as it's easiest -> Shouldn't do with big libraries

from random import *
Translates to -> from the random library import everything

Organizing our code with Modules

File called: functions.py

```
def print_name(name):
    print("Name:", name)

def print_age(age):
    print("Age:", age)

def print_country(country):
    print("Country:", Country)
```

Main file -> main.py

```
from functions import *

print("Hello, Welcome!")

name = input("Name: ")
age = input("Age: ")
country = input("Country: ")

print_name(name)
print_age(age)
print_country(country)
```

We store all of our functions in a **separate file**. We then can **import** them to use throughout our **main** python file

This allows for cleaner code and more readable

Working with Random Module

Get the computer to generate random numbers

Random Module

Random Functions	What they do
random( )	Generate a random number (0 - 0.9999)
randint(**5** , **10**)	Generates a random number from **5** - **10**
shuffle(something)	Used to mix up/shuffle a List

```
random_number = randint(0,100)
print(random_number)
```

#Ouput -> 56

These are some of the most popular functions from the random module

Importing random module

*We introduced how to import in our last lesson & the 3 different styles with their benefits

import random	geuss = random.randint(5 , 10)
from random import randint	geuss = randint(5 , 10)
from random import *	geuss = randint(5 , 10)

In example 1, you **must** tell python from which module the function is coming from, which can lead to more code, but also help understand your code better

Using randint() with our code

Example: An Airline needs to decide which flight to book overbooked passengers on. The program is **randomly** assigning **one of three** flights to each guest

```
from random import randint

passenger = input("Enter passenger name (quit to quit): ")

while passenger != "quit"':

    flight_number = randint(1, 3)

    print(passenger , ", Flight Number:" , flight_number)

    passenger = input("Enter passenger name (quit to quit): ")
```

Using randint() with our code

Example: An Airline needs to decide which flight to book overbooked passengers on. The program is **randomly** assigning **one of three** flights to each guest

```
from random import randint

passenger = input("Enter passenger name (quit to quit): ")

while passenger != "quit":

    flight_number = randint(1, 3)

    print(passenger , ", Flight Number:" , flight_number)

    name = input("Enter passenger name (quit to quit): ")
```

We **import** randint to use

Loop runs **until** "quit" is entered

flight_number is a variable holding a **random** number between **1 - 3**

Prints off the passenger name with a random flight number

Working with Time Module

Creating a Stopwatch or a Countdown

Popular Time Functions

Time Functions	What they do
time()	Return the number of seconds since the beginning of the epoch (January 1, 1970)
sleep(number)	Pauses/Delays a program for a number of seconds

If you print of “time” it will return the number of seconds that has passed since Jan 1, 1970. As of today that is **1,680,158,034.534593**

How can you use time()

Here is code used to create a simple Timer with the time module

```
from time import *
stopwatch = input("1 - start, 0 - stop:")
while stopwatch != "0":
    if stopwatch == "1":
        start_timer = time()
    stopwatch = input("0 - End Timer: ")
end_timer = time()
total = end_timer - start_timer
print("Total time:", total , "sec")
```

We create a **variable to hold time**. When Python reaches this point, the **clock begins**.

Note: The timer begins from the current time: **1,680,158,034.534593**

We create a **variable to hold time**. When Python reaches this point, the **clock ends**. The variable now holds the new value from the time passed

How can you use time()

```
from time import *
stopwatch = input("1 - start, 0 - stop:")
while stopwatch != "0":
    if stopwatch == "1":
        start_timer = time()
    stopwatch = input("0 - End Timer: ")
end_timer = time()
total = end_timer - start_timer
print("Total time:", total, "sec")
```

If the timer begins counting at **1,680,158,034.534593**

We allow the timer to run for 30 seconds

The value of end_timer will be **1,680,158,064.534593**

In order to find the time that has passed we need to **subtract end_timer from start_timer**.
This would be **1,680,158,064.534593 - 1,680,158,034.534593 = 30** (30 seconds)

Rounding down numbers

We need to be able to deal with **long decimal numbers**, most of the time we don't need more than 1 or 2 decimals

total_time = 1,680,158,034.534593 ← We need to shave some decimals

updated_time = round(total_time, 2)

We can used a built-in Python function called round() to round a number down

round(variable , number of decimals you want)

What do you want to round down?

An **integer**, representing the number of decimals you want after rounded down

Rounding down numbers

We need to be able to deal with **long decimal numbers**, most of the time we don't need more than 1 or 2 decimals

total_time = 1,680,158,034.534593

```
updated_time = round(total_time, 3)

print(updated_time)
```

Output in console
1,680,158,034.**534**

```
updated_time = round(total_time, 1)

print(updated_time)
```

Output in console
1,680,158,034.**5**

Objects in Python
Object Oriented Programming

There are 4 people in a family, They all relate to each other but are special in their own way

What is an Object?

An Object is **a single similar item** from the same **group**, but each object has its own **properties** and **special characteristics**

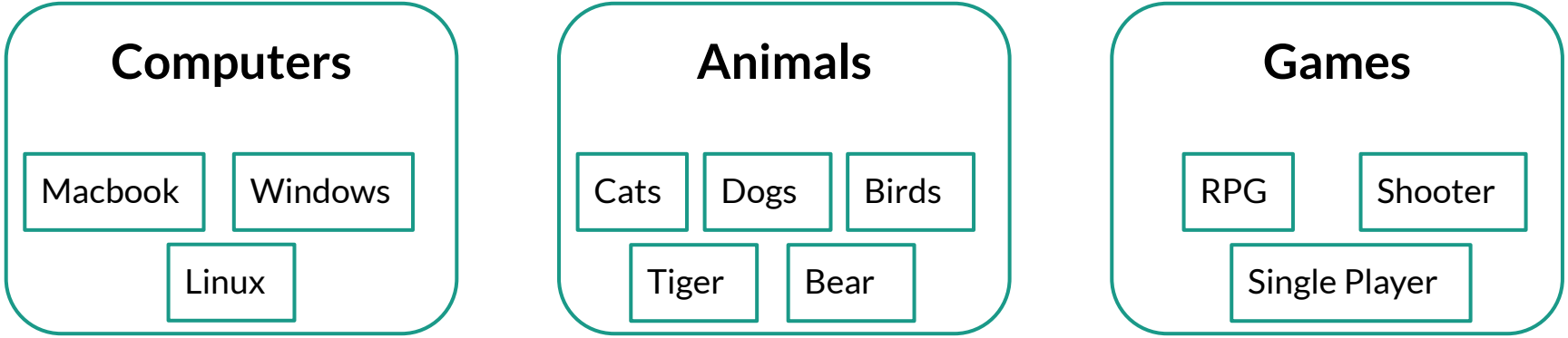
Car

-4 Wheels
-2 Seats
-Blue
-Convertible

-4 Wheels
-2 Seats
-Yellow
-Luxury

-4 Wheels
-2 Seats
-Black
-Vintage

Object Examples



Example: Animals -> **Animals is a Class** (Parent) and Cat, Dog, Etc are **all Objects**. While they all come from the **same parent**, they each have a **unique** name and attributes.

While they <u>may share</u> similar properties, such as eyes, fur, etc. They all have <u>special properties</u> that nothing else has

Working with and creating Objects

Each object has **properties** and is controlled by **methods**

Property -> A variable -> Specifically a **Variable in a Class**
Method -> A function -> Specifically a **Function in a Class**

Property Examples	Method Examples
airplane.speed = 500	airplane.fly()
car.speed = 50	car.drive()
boat.speed = 35	boat.sail()

*A <u>method</u> **must be** linked to an <u>Object</u> in order to use it

Working with and creating Objects

Each object has **properties** and is controlled by **methods**

Property -> A variable -> Specifically a **Variable in a Class**
Method -> A function -> Specifically a **Function in a Class**

Property Examples	Method Examples
airplane.speed = 500	airplane.fly()
car.speed = 50	car.drive()
boat.speed = 35	boat.sail()

*A method and property **must be** linked to an Object in order to use it

How to create & control objects?

There are 9 Turtles

Although they are all from the <u>same family</u> **(class)**, They all have a <u>different appearance</u> (**property**) and <u>different action</u> (**method**)

Property -> turtle.color = red
Method -> turtle.move(20)

Creating an Object

*An Object / Instance **is equal to** a Class

```
black_turtle = Turtle()
black_turtle.color("black")
black_turtle.forward(50)
```

An object called black_turtle

This is a Class called Turtle. A class is **always capitalized** and **ends with** parentheses

Creating a Property for my Turtle and setting its value to be black

Giving my Turtle something to do (Method)

Working with the Turtle Module

Before we look at Object Oriented Programming it's good to have a decent understanding of **Objects, Properties, Methods and Classes**

```
from turtle import *
```

Importing the Turtle Module

```
red_turtle = Turtle()
red_turtle.color("red")

purple_turtle = Turtle()
purple_turtle.color("purple")

orange_turtle = Turtle()
orange_turtle.color("orange")
```

Creating 3 different turtles (objects) and giving them each a different color (property)

Properties & Methods in Turtle

Properties / Methods in Turtle	Explanation
turtle.speed = 6	Set the speed of a turtle
turtle.color("blue")	Set the color of a turtle
turtle.shape("turtle")	Set the shape (turtle, circle,triangle, square, arrow)
turtle.width(5)	Set the line thickness
turtle.forward(100)	Method to move the turtle forward
turtle.left(90) or turtle.right(90)	Method to turn the turtle (degrees)

Changing the location

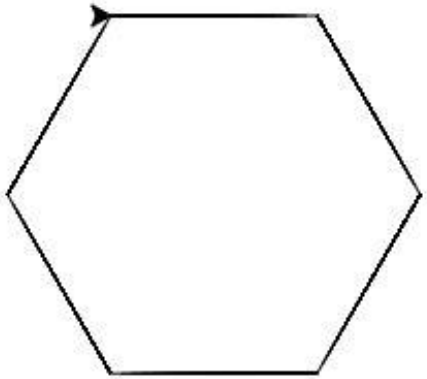
Methods for changing position	Explanation
turtle.penup()	Pick up the turtle off the paper
turtle.goto(x,y)	Go to a new position
turtle.pendown()	Put the turtle back onto the paper

Steps:

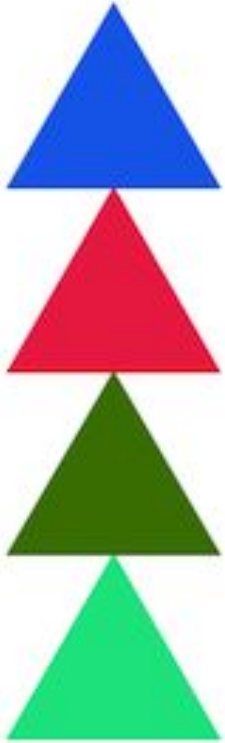
1. Create an Object
2. Set the Object properties
3. Change the Objects' Position
4. Give your object commands to move

Challenges for Turtle

1. Draw a Hexagon
2. Draw a Hexagon with only 3 lines of code (for loop)
3. Draw 3 Triangles at the same time in different locations
4. Program two functions that your Turtle can use
5. Use a Conditional Statement to draw something based on an input



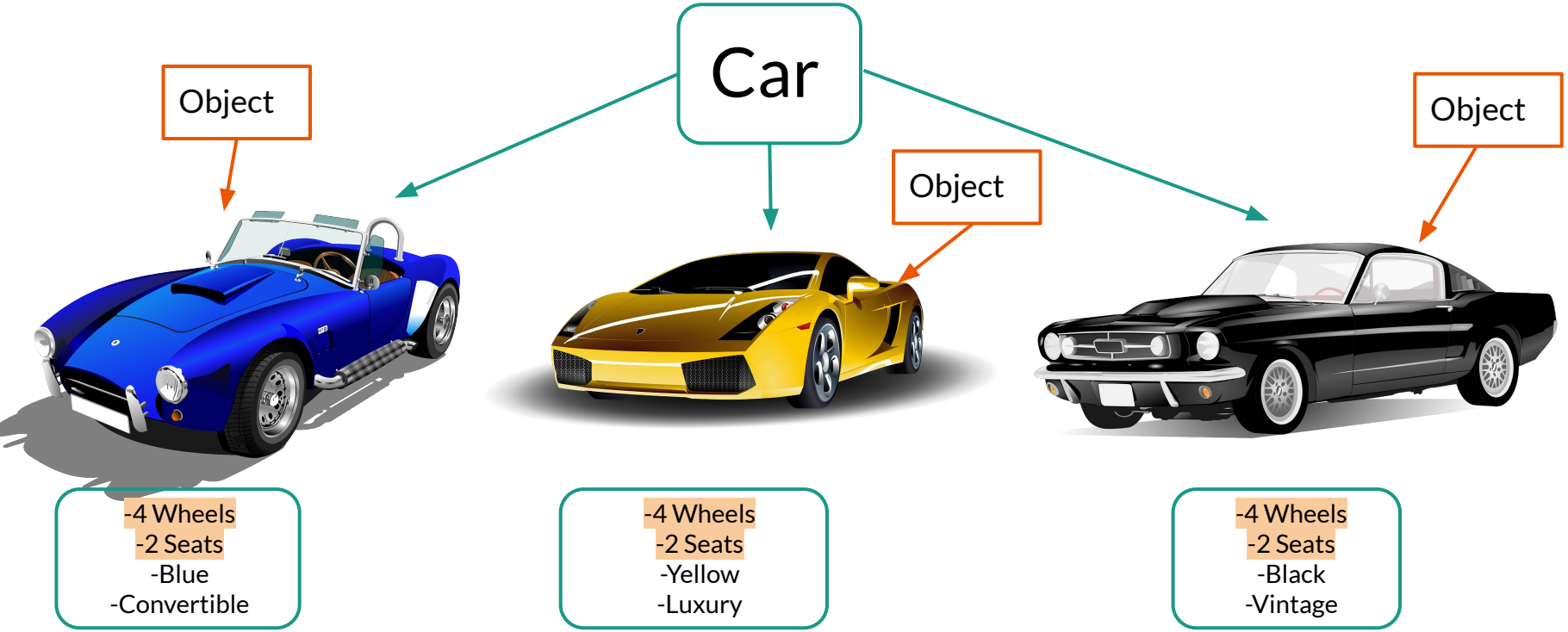
begin_fill()	Start to fill a solid color
end_fill()	End the fill of a solid color
exitonclick()	Allows your image to stay on the screen once finished



Creating your own Classes & Objects

A class is like a Family, They share some of the same properties but also have their own unique properties

Class -> is a name for similar/many objects



An Object (instance) of a Class

instance = Class()

Object

Object is a Variable equal to a class

A Class is **always** Capitalized and followed by parentheses

red_turtle = Turtle()

Object

Class

The Constructor Function

It's a method (function) that is **automatically ran** when an object is created. It creates an instance of a specific class.

Constructor

↓

Construction

↓

Used to Build

The Constructor Function

It's a method (function) that is **automatically ran** when an object is created. It creates an instance of a specific class.

__init__()

This is the name **we must** use for the Constructor Function

__init__ -> Initialize -> Start

Constructor

↓

Construction

↓

Used to Build

Building a Class

*For every parameter in __init__ **you must** also give the same number of Arguments when you create an object

```
class  Class name ():

    def __init__(self, parameter ):
        self. property = parameter

    def class_details(self):
        print("Show me the details:", self. Property)

object = Class_name ( Argument )
```

class allows us to define/create a new class

self is **always** the first parameter, it allows us to use properties and methods anywhere within a class

Creating an instance of the class.
The Arguments are given to the __init__ function

Building a Class

*For every parameter in __init__ **you must** also give the same number of Arguments when you create an object

```
class Class name ():

    def __init__(self, parameter ):
        self. property = parameter

    def class_details(self):
        print("Show me the details:", self. Property)

object = Class_name ( Argument )
```

A variable in a class

These are **Methods**, remember a method is just a **function in a class**

Creating an instance of the class. The Arguments are given to the __init__ function

Building a Class

Here we create a **class Agent**

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car
```

This is our **constructor** function (__init__). This is where we **create all the variables** (properties) our class will need to run

We create **two unique objects**, each object has the **value of Player** Class

```
spy = Agent("James Bond", 100, "Jaguar")
villain = Agent("Ethan Hunt", 40, "Ferrari")
spy.player_info()
villain.player_info()
```

Each object has **3 arguments**, these are passed to our **__init__** method to use throughout our class

Building a Class

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car

    def player_info(self):
        print("Welcome, ", self.name)
        print("Your health: ", self.health)
        print("Car Choice: ", self.car)

spy = Agent("James Bond", 100, "Jaguar")
villain = Agent("Ethan Hunt", 40, "Ferrari")
spy.player_info()
villain.player_info()
```

Here we create a **class Agent**

self.car now holds the value of car. Whatever we pass into our car **parameter** from our **arguments** will be the value

We can use properties just like we do Variables

self is like a **key**. It unlocks all the properties & methods to allow us to use them anywhere throughout our Class

Essentially turns everything "**Global**" within the Class

Building a Class

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car

    def player_info(self):
        print("Welcome, ", self.name)
        print("Your health: ", self.health)
        print("Car Choice: ", self.car)

spy = Agent("James Bond", 100, "Jaguar")
villain = Agent("Ethan Hunt", 40, "Ferrari")
spy.player_info()
villain.player_info()
```

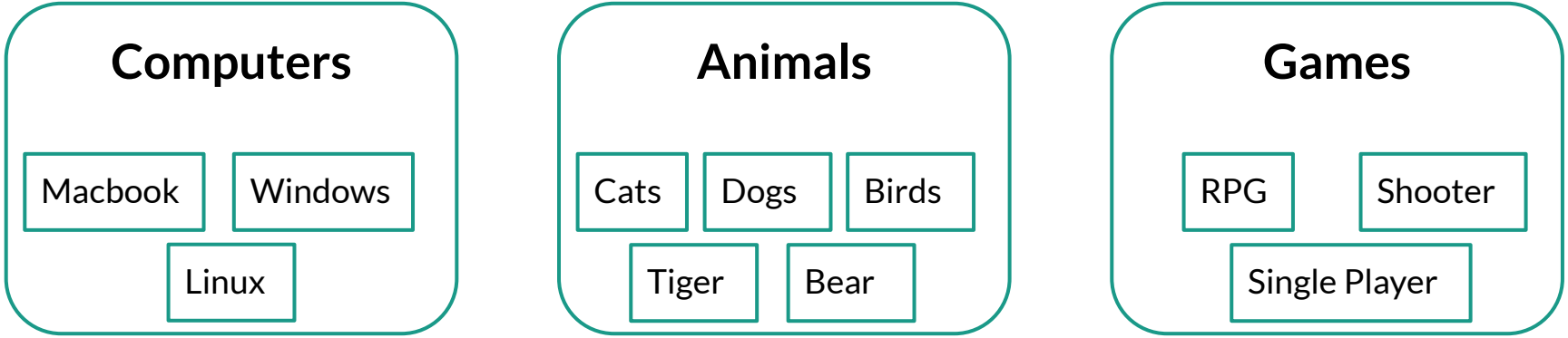
Output in console
Welcome, James Bond Your Health: 100 Car Choice: Jaguar Welcome, Ethan Hunt Your Health: 40 Car Choice: Ferrari

Here we **link our methods** from our class to our objects, **spy & villain**

Superclass and Inheritance

Who doesn’t love Inheritance? What do you think of when you hear the word, Inheritance?

Superclass & Inheritance



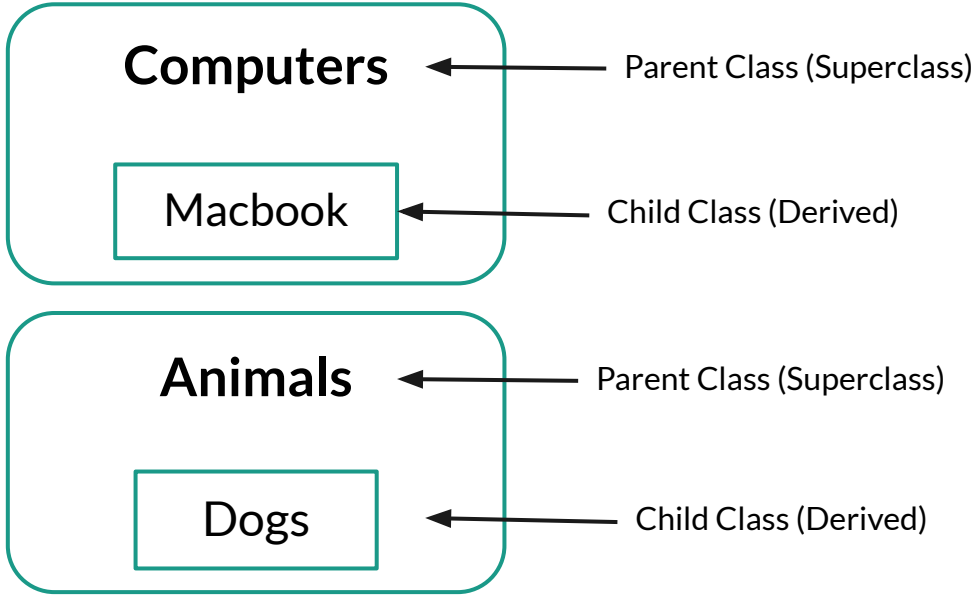
Example: Animals -> **Animals is a Class** (Parent) and Cat, Dog, Etc were all objects. We could create a class for each animal as well, taking key data from the parent

Quick Flashback to our objects lesson

Superclass a.k.a Inheritance

Many times we need to transfer the properties/methods from one class to another class.

This is known as **inheritance**



We can create one main class that all other classes can use.

*The **children inherit from the parents** but also have attributes and actions of their own!

Superclass & Derived Classes

All Ferraris **are** cars

All sofas **are** furniture

All dogs **are** animals

All jeans **are** pants

Types of Inheritance	Notes about Inheritance
The child class is given new actions (methods) but **new properties** are not given	We **do not** need to create a constructor (__init__) within the child class. **We can use the parent** __init__
The child class is given **new properties** and **new methods**	We need to create a constructor (__init__) within the child class, and also activate that parent constructor

Creating a child class (derived class)

Creating a child class that **only needs new methods** not properties

When we make an instance of a child class, the superclass constructor (__init__) will be **called and used in the derived class**

```
class Child_class( Superclass_name ):
    def method_one(self, parameter_1):
        This code will run with the object & properties
    def method_two(self, parameter_1):
        This code will run with the object & properties
```

Creating a child class with new properties

The constructor in a child class **takes the properties of the superclass** and any **new properties we create**

```
class Child_class(Superclass_name):
    def __init__(self, parameter_1, parameter_2):
        super().__init__(parameter)
        self.property = parameter
    def method_one(self, parameter):
        This code will run with the object & properties
```

super() allows us to **inherit all the properties and methods** from the superclass (parent class)

When we create an object of the child class, the superclass constructor (__init__) is **automatically called** and used.

How inheritance works in our code

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car

    def player_info(self):
        print("Welcome, ", self.name)
        print("Your health: ", self.health)
        print("Car Choice: ", self.car)
```

```
class Spy( Agent ):
    def spy_talk(self):
        print("My name is," , self.name)

    def shoot(self, bad_guy):
            if bad_guy.health > 0:
                    Bad_guy.health -= 20
            elif bad_guy.health <= 0:
                    print(self.name, "has assassinated" , bad_guy.name)
```

*We created a child class called "Spy". This will be a class only used by our good guy. We **inherit all the properties from our superclass**, which we **can use throughout our child class**

Inheritance break down

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car

    def player_info(self):
        print("Welcome, ", self.name)
        print("Your health: ", self.health)
        print("Car Choice: ", self.car)
```

```
class Spy( Agent ):
    def spy_talk(self):
        print("My name is," , self.name)

    def shoot(self, bad_guy):
        if bad_guy.health > 0:
            bad_guy.health -= 20
            print(bad_guy.name, "has lost", bad_guy.health)

james_bond = Spy("James Bond", 100, "Jaguar")
ethan_hunt = Agent("Ethan Hunt", 50, "Ferrari")
james_bond.player_info()

james_bond.shoot(ethan_hunt)
```

We will give an object as an argument when we call this method

We give ethan_hunt as the bad_guy argument, this will result in him **losing 20 from his health** when we **call the method shoot()**

*__Never__ use self outside of the class

2nd route for Inheritance

When and How to use the super() function

Route 2 for Inheritance

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car

    def player_info(self):
        print("Welcome, ", self.name)
        print("Your health: ", self.health)
        print("Car Choice: ", self.car)
```

```
class Spy( Agent ):
    def __init__(self,name, health, car, location ,agency):
        super().__init__(name, health, car)
        self.location = location
        self.agency = agency

    def spy_talk(self):
        print("My name is," , self.name)
        print("I am based at," , self.agency , "in" , self.location)
```

We are **initializing the properties from the Superclass** by using the **super()** function as well as creating **2 new properties** for the child class.

How to use Route 2 for our Inheritance

```
class Agent():
    def __init__(self, name, health, car):
        self.name = name
        self.health = health
        self.car = car

    def player_info(self):
        print("Welcome, ", self.name)
        print("Your health: ", self.health)
        print("Car Choice: ", self.car)
```

```
class Spy( Agent ):
    def __init__(self, name, health, car, location ,agency):
        super().__init__(name, health, car)
        self.location = location
        self.agency = agency
    def spy_talk(self):
        print("My name is," , self.name)
        print("I am based at," , self.agency , "in" , self.location)
```

```
james_bond = Spy("James Bond", 100, "Jaguar", "London", "MI6")
james_bond.spy_talk()
```

Output in console
My name is, James Bond I'm based at MI6 in London

*Never use self outside of the class

Working with Lists

A list simply put is a list of things!

Looking at a list

A way to **store ordered data of different data types**

ages = [25, 43, 57, 18, 34, 16, 20]

Creating a new list:

- You can use the square **brackets** -> names = []
- You can use the **list function** -> names = list()

*Declares an empty list

What is a List & what a List looks like

A way to **store ordered data of different data types**

Example: Here is a list of different ages

ages = [25, 43, 57, 18, 34, 16, 20]

Variable holding a list

Elements in a list

element	25	43	57	18	34	16	20
index	0	1	2	3	4	5	6

The position **[index]** of the elements in our list

*Remember pythons starts on 0

If we want a **certain element** from the list we need to **get the position of the item** in the list

ages[3] -> 18
ages[1] -> 43
ages[-1] -> 20

We use the brackets [] to justify the position (index) in a list

List code example

```
ages = [18, 34, 55, 23]
age = int(input("Enter an age: "))
while age != 0:
    if age in ages:
        print("You have already entered that age")
    else:
        ages.append(age)
    age = int(input("Enter an age: "))
ages.sort()
print("List of ages in your list:", ages)
```

Example: We create a program to loop through to allow us to **add multiple ages to our list**. We check **if the age we entered is already in our list**, if it is not, then we want to **add the age entered to our list**. Once the program ends we **automatically sort** our list from smallest to greatest.

List code breakdown

```
ages = [18, 34, 55, 23]
age = int(input("Enter an age: "))
while age != 0:
    if age in ages:
        print("You have already entered that age")
    else:
        ages.append(age)
    age = int(input("Enter an age: "))
ages.sort()
print("List of ages in your list:", ages)
```

We have a list named ages

Checking **if age** (input) is **inside our list** (ages)

We **add our input** (age) **to our list** (ages)

We **automatically sort** our list

in	Allows us to <u>check **if** something</u> is **in** something else
.append()	Append means add, we can **add an item to the list**
.sort()	**Automatically sorts our list**, A-Z or 1-10,000

For loops with lists

```
ages = list()
age = int(input("Enter an age: "))
minors = 0
seniors = 0
while age != 0:
    for age in ages:
        if age < 18:
            minors += 1
        elif age >= 70:
            seniors += 1
        ages.append(age)
    age = int(input("Enter an age: "))
print("All the ages:", ages)
print("Number of Minors:", minors, "- Number of Seniors:", seniors)
```

For loops become incredibly powerful to **allow us to go through each element in a list** to **search for something**

Counter Variables

Literal Translation -> **for every age in my list ages**

For loops allow us to **use the current index** (age) in our conditions to check against something

For loops with Lists

```
ages = list()
age = int(input("Enter an age: "))
minors = 0
seniors = 0
while age != 0:
    for age in ages:
        if age < 18:
            minors += 1
        elif age >= 70:
            seniors += 1
        ages.append(age)
    age = int(input("Enter an age: "))
print("All the ages:", ages)
print("Number of Minors:", minors, "- Number of Seniors:", seniors)
```

For loops become incredibly powerful to **allow us to go through each element in a list** to **search for something**

Counter Variables

Literal Translation -> **for every age in my list ages**

Allows us to **use the current index** (age) in our conditions to check against something

Generate a List from a single input

Split an input into a list without looping

Using the .split() method

.split() allows us to literally **split a variable** and **use each piece to generate a new list**

```
alphabet = "A B C D E"

alphabet = alphabet.split(" ")

print("List of letters:" , alphabet)
```

We have a variable, its value is a string with spaces

Alphabet is now equal to alphabet split anywhere a space is found

Output in console
List of letters: ["A" , "B" , "C" , "D" , "E"]

Split() breakdown

alphabet = “A B C D E”

alphabet = alphabet.split(“ ”)

split() takes 1 argument, we tell python “where” we want the split to occur

In the first example we have a space, python will split when it finds a space

binary = 00010000100001

binary = binary.split(1)

In this example we have the number 1, python will split when it finds a 1

print(binary) -> [000,0000,0000]

Code Breakdown

```
ages = int(input("Enter ages with a space: "))
ages = ages.split(" ")
seniors = 0
minors = 0
for age in ages:
    if age >= 70:
        seniors += 1
    elif age < 18:
        minors += 1
print("List of all ages:", ages)
print("-Seniors:", seniors, "-Minors:",minors)
```

Here we create a list called ages by splitting our input.

We use our index (age) from our for loop, everytime the loop runs, it checks if that index is a senior or minor

Similar to our original example

Key Takeaway points

names = [ ]	Creates a list, **empty or not**
names = list()	Creates a new **empty** list
.split()	**Splits** a variable and **creates** a list
.append()	**Add** an item to a list
.remove()	Can **remove** an item from a list
.sort()	Automatically **sorts** a list
names[3]	Gets the 4th element from a list
names[-1]	Gets the last element from a list

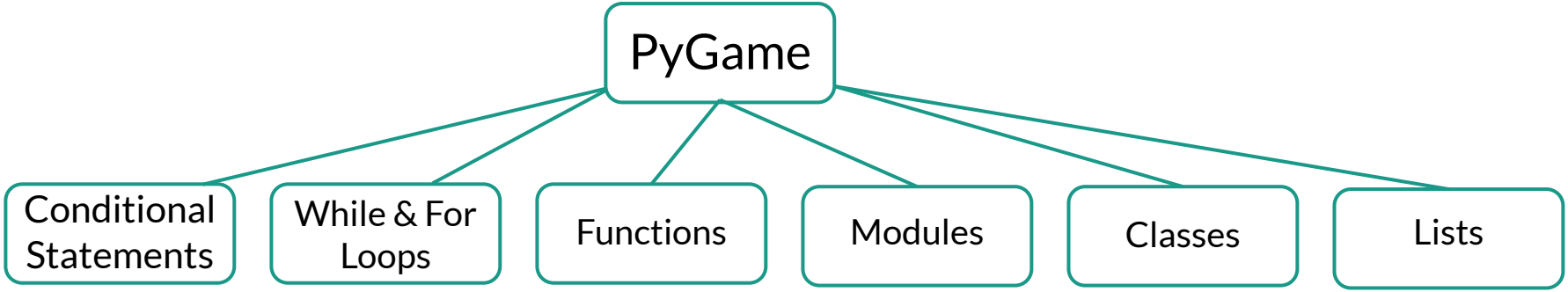
Welcome to PyGame

Creating 2D retro games with Python. Why will we do Pygame?

Why Pygame?

This course was created to build you a **strong foundation in Python**. If we don't use and reinforce these foundational skills, more advanced Python topics will become a struggle

PyGame is one of the best ways possible to put your knowledge to the test! We will **use everything we have learned** throughout this course in a single application



Basic PyGame Setup

- Importing needed modules ← Keeping our imports at the top of our code
- Creating Game Variables ← What variables will we need?
- Game Settings ← Creating a game screen, timer and name
- Creating Classes ← Superclass with Derived classes for our objects
- Main Game Loop ← Every game needs a game loop, end of the code
- Game Events ← Events -> Keypress, Click, Drag, Key release

Creating our a game screen - Set up

Most basic setup to get our game screen

```
import pygame
pygame.init()

screen = pygame.display.set_mode((600, 400))
pygame.display.set_caption("My Game Name")
clock = pygame.time.Clock()
background = transform.scale(image.load("back.png") , (600,400))

run = True
while run:

        screen.blit(background , (0,0))

        pygame.display.update()
        clock.tick(FPS)
```

We must **initialize** the PyGame Module

set_mode() allows us to **set the width and height** of our game window, it's **linked to the display**

Creates a **clock** we can use for **FPS**, How many times our game <u>will run everytime our loop runs</u>

Updates the screen as the loop runs

Add Events to a game

Types of events on the computer: **Keypress, Key release, Click, Drag, Quit**

The first event any game checks for is, **quit**.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        quit()
        run = False
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_SPACE:
            #Do This
```

QUIT
KEYDOWN
KEYUP
MOUSEBUTTONDOWN
MOUSEBUTTONUP
MOUSEMOTION

K_LEFT
K_RIGHT
K_1
K_4
K_a
K_d

*Goes at the top of your game loop

Code used with Pygame

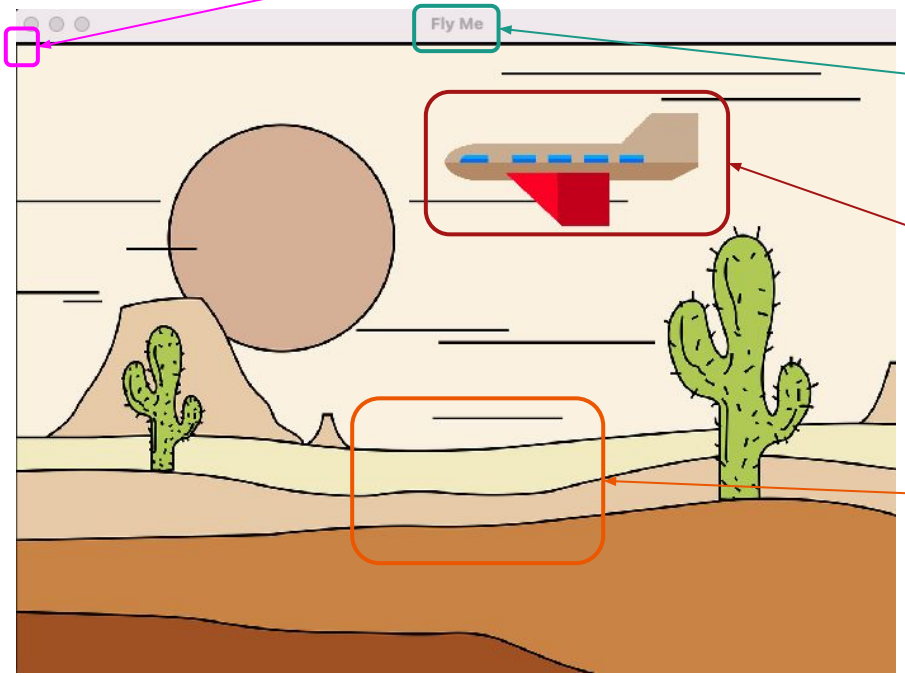
Syntax	What it does
screen = display.set_mode((700, 500))	Create the **screen size** (width, length)
clock = time.Clock() clock.tick(60)	This Clock object **keeps track of time** and is used for our game **FPS**
display.set_caption("My Game Name")	Set the screen title.
background = image.load("background.jpg") **(or)** background = transform.scale(image.load("background.jpg") , (700, 500))	Creating a picture object, **change the picture size** to the screen width / height
screen.blit(background,(0, 0))	Display the background picture on the screen **blit** means "**Paste**" in Pygame

Continued code used with pygame

Syntax	What is does
keys = key.get_pressed()	Returns the current state of the keys (**Keydown = True** otherwise it returns False
if keys[K_UP]: player_y -= 15	**If the up arrow key is down**, decrease the Y position of the player by 15 pixels.
if keys[K_s] and player_y < 500: player_y += 15	If the "s" key is down **and** the bottom of the screen has not been reached, increase the Y position of the player by 15 pixels.
if keys[K_SPACE] and player_x > 0: player_x -= 15	If the "space bar" is down **and** the left of the screen has not been reached, decrease the X position of the player by 15 pixels.

Letter keys are lowercase and **arrow keys are uppercase**

Pygame Startup



X,Y (0,0) in Pygame is the top left of the screen

Has a game name of Fly Me

Moves with Arrow keys but **can not** leave the screen

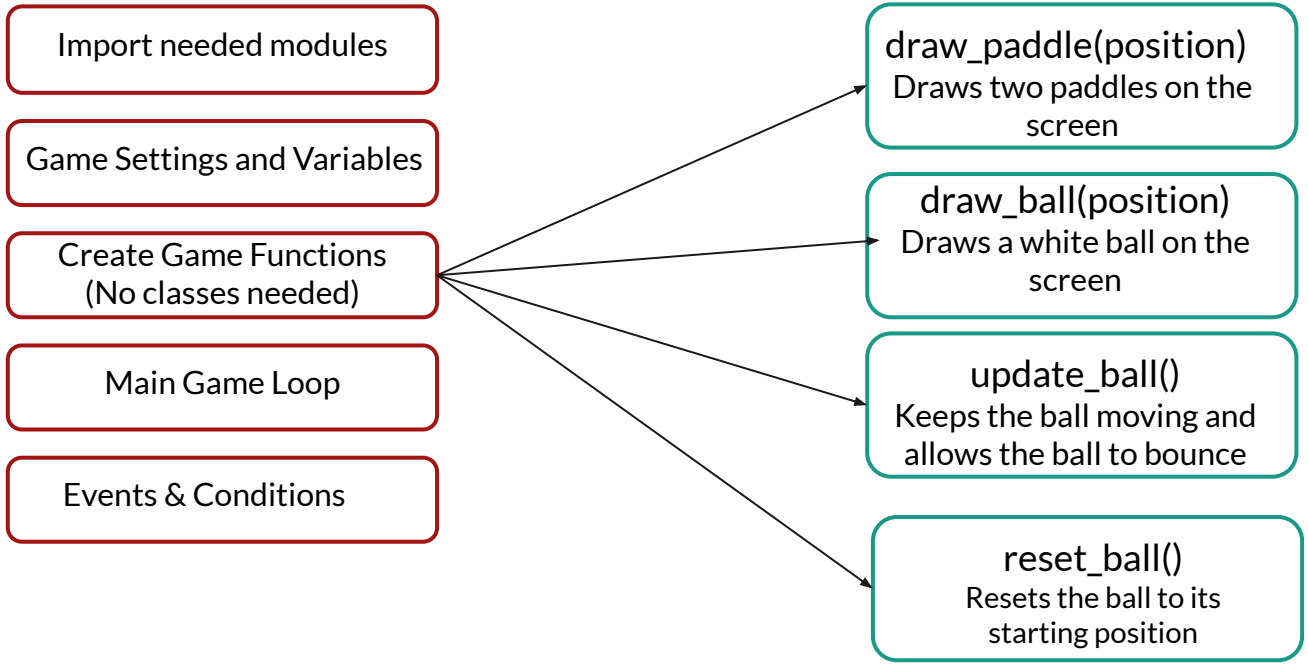
We paste the background in the first and center on the screen, done in our loop

*Use the files provided and try this quick setup on your own!

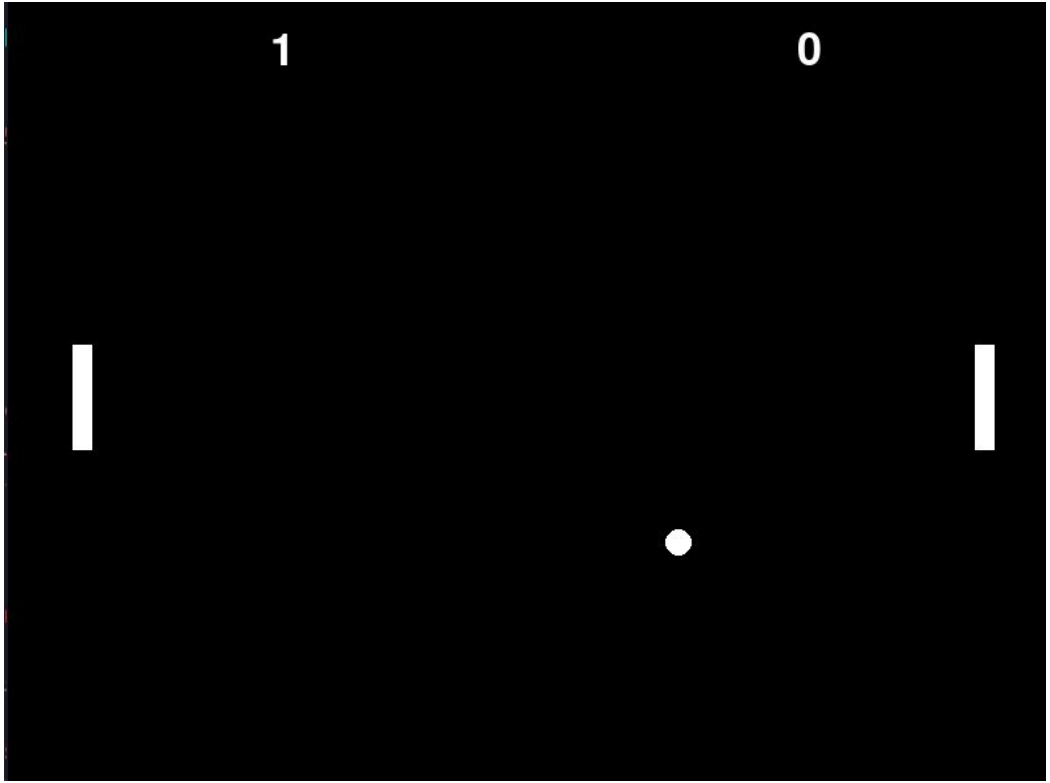
Retro Ping-Pong

Your first full fledged game!

Create your first game - Solo



Output of your Pygame



Hint - use the .fill() method to set the background color as Black

Creating Flappy Miner

How to create a Flappy Bird like game with our current knowledge?

Game Breakdown

When the **space bar is pressed**, our miner **jumps**, else he is falling

Every pipe that passes is at a **random position**

Our score increases as we go through each set of pipes

If you touch a pipe, our game ends

*Think about how you can use Lists

Class Breakdown

SuperClass

Constructor Method

Speed Property

Image Property

rect Property (.get_rect())

rect.x property

rect.y property

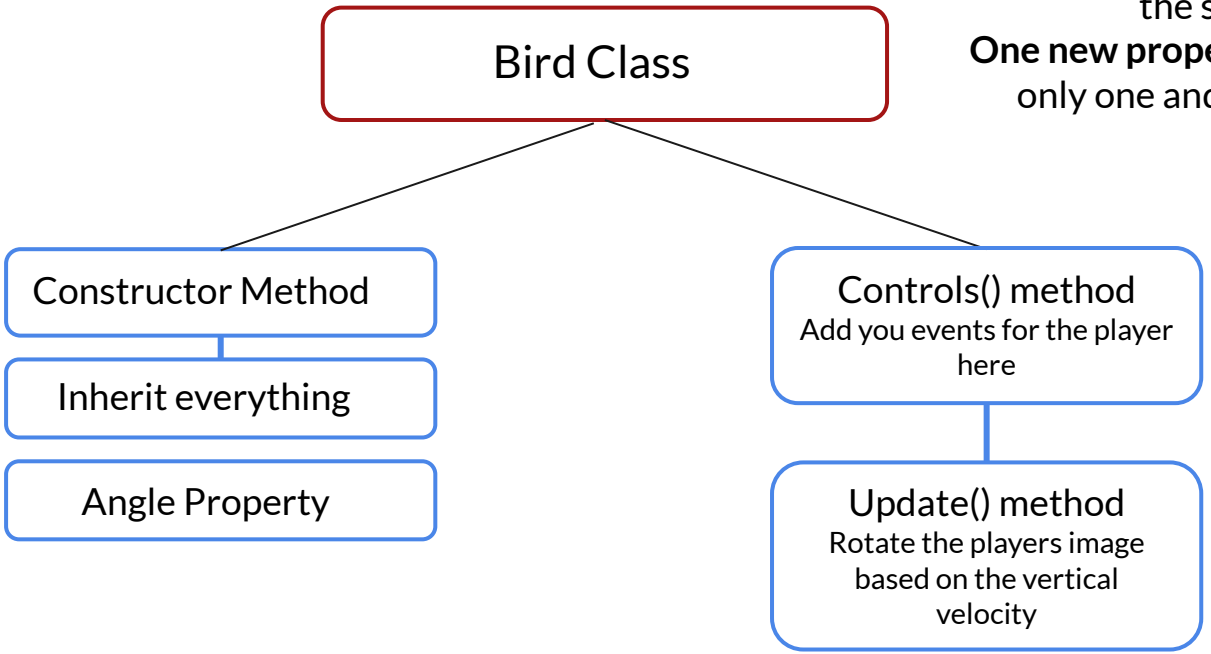
update() method
Will be used to .blit() all the images to the screen

Your parameters you may need are:

- Image
- X
- Y
- Speed

Bird Class Breakdown

This class will **Inherit everything** from the superclass.
One new properties will be needed, only one and one new method



Pipe Class

Note -> This class will not inherit from any other class

Pipe Class

- Constructor Method
  - Speed Property
  - Width Property
  - Height (randint)
  - top_rect (Rect())
  - bottom_rect (Rect())
- update() method
  Will be used to keep the top and bottom pipe in motion
- draw() method
  Will be used draw each white pipe onto the screen
  (draw.rect())
- reset() method
  Will be used to reset the pipes to a new position once they leave the screen

Bitcoin Maze

Collecting coins, avoiding ghosts, all to find the bitcoin wallet!

Game Breakdown

Use the **Arrow keys** to navigate through the maze

If the Miner touches the wall, they **go back to their starting position**

Collect the coins to increase your wallet size, **coins placed randomly**

If you **touch the ghost**, you go back to the starting position

The ghost is **always moving**

You must **collect all the coins and the BTC wallet** to win the game

Superclass Breakdown

The Superclass inherits **sprite.Sprite**

The sprite class is meant to be used as a base class for the objects in your game.

SuperClass

Constructor Method

Speed Property

Image Property

rect Property (.get_rect())

rect.x property

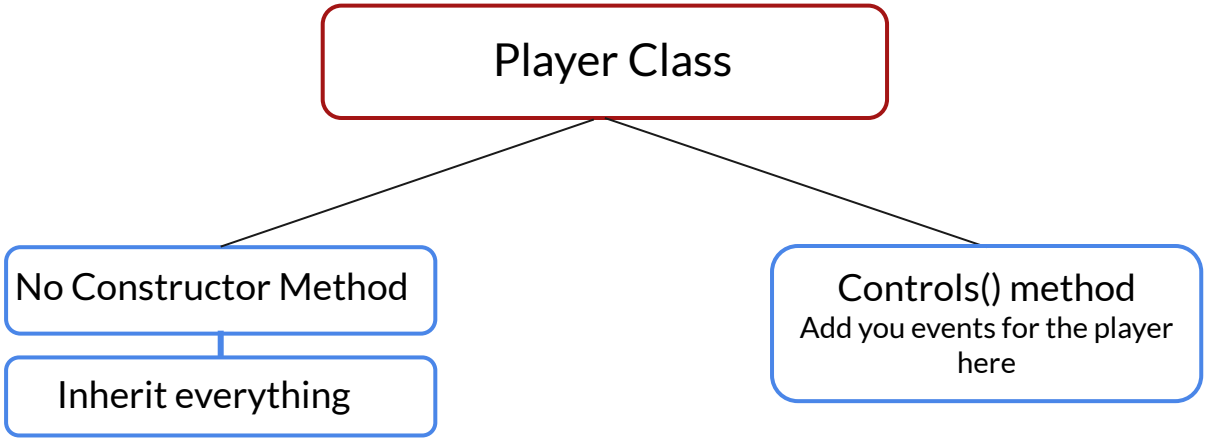
rect.y property

update() method
Will be used to .blit() all the images to the screen

Your parameters you may need are:

- Image
- X
- Y
- Speed

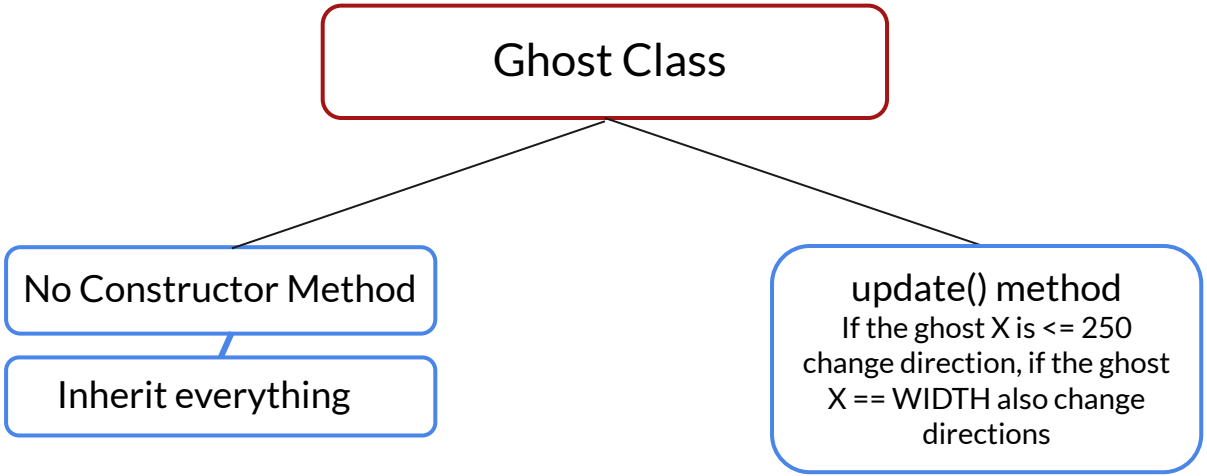
Player Class Breakdown



*__Arrow keys__ or __AWSD__ keys, your player can not leave the game screen

This class will __Inherit everything__ from the superclass.
__No new properties__ will be needed, only one new method

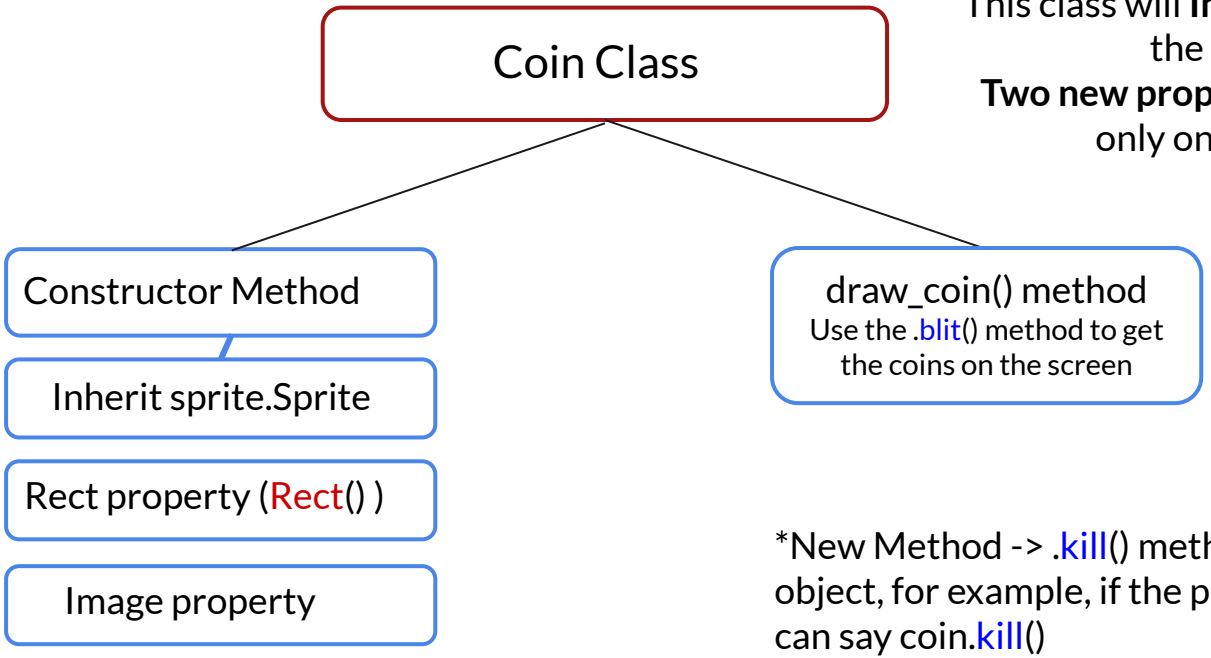
Ghost Class Breakdown



*Apply self.speed to the ghost in a condition to keep them in continuous motion

This class will **Inherit everything** from the superclass.
No new properties will be needed, only one new method

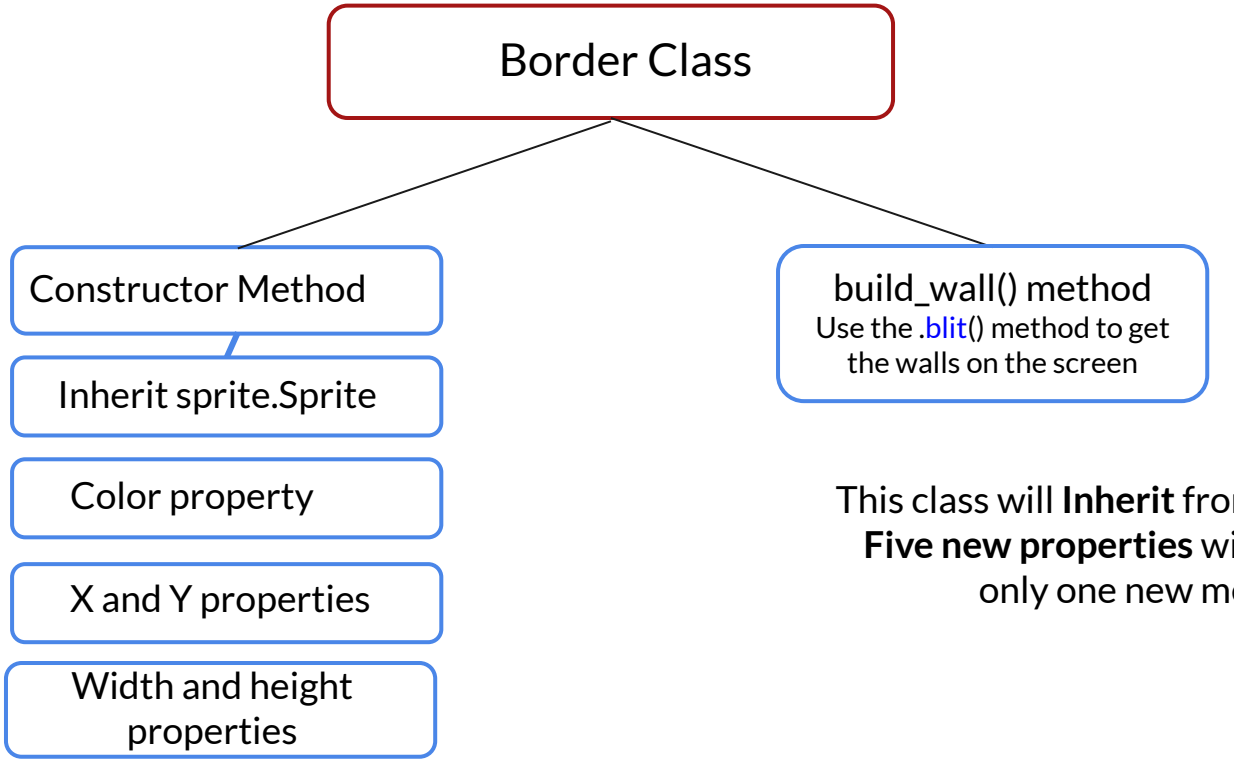
Coin Class Breakdown



This class will **Inherit everything** from the superclass.
Two new properties will be needed, only one new method

*New Method -> .kill() method will destroy an object, for example, if the player touches a coin you can say coin.kill()

Border Class Breakdown



This class will **Inherit** from sprite.Sprite
Five new properties will be needed,
only one new method

Working with sounds in PyGame

Syntax	What they do
mixer.init()	Allows the use of the Mixer
mixer.music.load("wave.mp3")	Loads background music
mixer.music.play()	Starts playing the background music
money = mixer.Sound("coins.mp3")	Load sound for a one-time playback (for example, when your player dies)
money.play()	Play the sound