Learn to Code with Python

1

Setup & Installation

hello_world.py

```
print("Hello world")
```

2

Welcome to Python

objects-strings.py

```
"sushi"
'sushi'
"warriorsfan123!"
"$5.00"

"You're right"
'Shakespeare once wrote "To be or not to be" in one of his plays'

" "
""
''

"5"
5

"""Hello
my name is
Boris
"""
```

the-print-function-I-single-argument.py

```
print("I love sandwiches")
print('I also love juice beverages')
print("We're working with Python")

print(5)
print(3 + 4)
print(10 - 8)

print("3 + 4")

print("3" + "4  ")
```

the-print-function-II-multiple-arguments.py

```python
print("A", "B")
print("C", "D", "E", "F")

print(5 + 3, 2 - 8)
print(5 + 3, 2 - 8, 3 + 1)

print("Hello", 10, 3.14)
```

the-print-function-III-parameters-and-arguments.py

```
print("ABC", "DEF", "XYZ")
print("ABC", "DEF", "XYZ", sep="!")
print("ABC", "DEF", "XYZ", sep="--*--")
print("ABC", "DEF", "XYZ", sep=" ")

print("ABC", "DEF", "XYZ", "!")

print("Hello", "Goodbye", end="!&*")
print("Hello")

print("A", "B", "C", sep="**", end="#")
print("A", "B", "C", end="#", sep="**")
```

comments.py

```python
print("Good morning")
#print("Good afternoon")
print("Good night")

# This will perform a mathematical calculation
print(1 + 1) # Adds together 1 + 1

# print(1 + 1)
# print(1 + 1)
# print(1 + 1)
```

3

Numbers, Booleans and Equality

mathematical-expressions.py

```
print(3 + 4)
print(3.5 + 6.3)
print(10 - 5)
print(5 * 3)

print("Ba" + "con")
print("lolo" * 32)

print(10 + 3.8)

print(5 ** 3)
print(4 ** 4)

print(3 + 5 * 3)
print((3 + 5) * 3)
```

division-floor-division-and-the-modulo-operator.py

```
print(15 / 3)
print(14 / 3)

print(14 // 3)
print(-14 // 3)

print(14 % 3)
print(15 % 3)
print(16 % 3)

print(1 % 3)
print(2 % 3)
print(3 % 3)
```

the-boolean-data-type.py (Part 1)

```
print(True)
print(False)
print("True")
print("False")

print(5 == 5)
print(5 == 8)
print(8.3 == 8.3)
print(8.3 == 4.1)
print(5 == "5")

print("ham" == "ham")
print("ham" == "bacon")
print("ham" == "Ham")
print("5" == "5")
print("" == "")
print("!==*" == "!==*")
```

the-boolean-data-type.py (Part 2)

```
print(5 == 5.0)
print(5 == 5.1)

print(True == True)
print(False == False)
print(True == False)

print(10 != 8)
print(10 != 10)
print("music" != "music")
print("music" != "noise")
print("music" != "Music")

print(10 != "10")
print(8.3 != 9.8)
print(3.0 != 3)
```

boolean-mathematical-operators.py

```
print(5 < 8)
print(10 < 7)
print(8 <= 8)
print(8 <= 11)
print(8 <= 7)

print(9 > 5)
print(10 > 20)
print(9 >= 9)
print(9 >= 5)
print(9 >= 10)

print(5 < 8 <= 10)
print(5 < 8 <= 7
```

the-type-function.py (Part 1)

```
print(type(5))
print(type(10))

print(type(3.8))
print(type(5.0))

print(type("computer"))
print(type("laptop"))
```

the-type-function.py (Part 2)

```python
print(type(5) == type(10))
print(type("computer") == type("laptop"))
print(type(5) == type(5.0))
print(type(5) == type("5"))

print(type(True))
print(type(False))
print(type(True) == type(False))

print(type([1, 2, 3]))
print(type({ "NJ": "Trenton" }))
```

type-conversion-with-the-int-float-and-str-functions.py

```
print(int(3.14))
print(int(3.99))
print(int("3"))
print(int(3))

print(float(5))
print(float("10.32"))
print(float(5.23))

print(str(5.35))
print(str(5))
print(str("Hello"))

print(str(5) + "Hello")

print(3 + 3)
print(4.3 + 5.7)
print(3 + 5.8)
```

4

Variables

intro-to-variables.py (Part 1)

```python
name = "Boris"
age = 27
handsome = True
favorite_language = "Python"

print(name)
print(handsome)
# print(occupation)

print(age + 4)
print("My name is", name, "and I am", age, "years old")

age = 23
print(age)

age = 27 + 10
print(age)
```

intro-to-variables.py (Part 2)

```python
fact_or_fiction = 6 < 10
print(fact_or_fiction)

chameleon = 5
print(chameleon)

chameleon = "Some string"
print(chameleon)

chameleon = 9.99
print(chameleon)

chameleon = False
print(chameleon)
```

multiple-variable-assignments.py

```
a = 5
b = 5

a = b = 5

b = 10
print(b)
print(a)

a = 5
b = 10

a, b = 5, 10

a, b, c = 5, 10, 13
print(a)
print(b)
print(c)
```

augmented-assignment-operator.py (Part 1)

```python
a = 1
a = a + 2
print(a)

a = 1
a += 2
print(a)

word = "race"
word = word + "car"
print(word)
```

augmented-assignment-operator.py (Part 2)

```
word = "race"
word += "car"
print(word)

b = 5
b = b * 3
print(b)

b = 5
b *= 3
print(b)

# -=
# /=
```

user-input-with-the-input-function.py

```python
# first_name = input("What's your first name? ")
# print("It's nice to meet you,", first_name)

# Prompt the user for two numbers, one at a time
# The numbers will be received as strings
# Convert both numbers to integers
# Print a message that includes the sum of the two numbers
print("Let me help you add 2 numbers")
first_number = int(input("Enter your first number! "))
second_number = int(input("Enter your second number! "))
print("The total is", first_number + second_number)
```

the-NameError-ValueError-TypeError-and-SyntaxError-Exceptions.py

```python
print(type(5))
print(type(10))

print(type(3.8))
print(type(5.0))

print(type("computer"))
print(type("laptop"))
```

5

Functions

intro-to-functions.py

```python
def output_text():
    print("Welcome to the program!")
    print("It's nice to meet you")
    print("Have an excellent day programming!")

output_text()
output_text()
```

parameters-and-arguments.py

```
def p(text):
    print(text)

p("Hello")
p("Goodbye")
p("OK")
# p()

def add(a, b):
    print("The sum of", a, "and", b, "is", a + b)

add(3, 5)
add(-4, 7)

# add()
# add(1)
add(3, 5, 7)
```

positional-arguments-and-keyword-arguments.py

```python
def add(a, b, c):
 print("The sum of the three numbers is", a + b + c)

add(5, 10, 15)
add(a = 5, b = 10, c = 15)
add(5, b = 10, c = 15)
add(5, c = 15, b = 10)
```

return-values.py

```python
def add(a, b):
    return a + b
    print("This is nonsense")

result = add(3, 5)
print(result)
```

default-argument-values.py

```python
def add(a = 0, b = 0):
    return a + b

print(add(7, 8))
print(add(10))
print(add())
```

the-none-type.py

```python
a = None
print(type(a))

def subtract(a, b):
    print(a - b)

result = subtract(5, 3)
print(result)
```

functions-annotations.py

```
def word_multiplier(word: str, times: int) -> str:
    return word * times

print(word_multiplier(10, 5))
```

6

Strings: The Basics

length-concatenation-and-immutability.py

```
print(len("Python"))
print(len("programming"))
print(len("!@#$"))
print(len(" "))
print(len(""))

# print(len(4))
# print(len(3.14))

print("Boris" + "Paskhaver")
print("Boris " + "Paskhaver")
print("Boris" + " Paskhaver")
print("Boris" + " " + "Paskhaver")

print("a" "b" "c")

print("---" * 30)
```

string-indexing-with-positive-values.py

```
best_language_ever = "Python"

print(best_language_ever[0])
print(best_language_ever[1])
print(best_language_ever[3])
print(best_language_ever[2 * 2])

print(len(best_language_ever))
print(best_language_ever[5])

# print(best_language_ever[100])

# best_language_ever[0] = "B"
```

string-indexing-with-negative-values.py

```python
topic = "programming"

print(topic[-1])
print(topic[-2])
print(topic[-5])

# print(topic[-100])
```

string-slicing-I-slicing-by-range.py (Part 1)

```python
address = "Attractive Street, Beverly Hills, CA 90210"

print(address[0:3])
print(address[0:4])
print(address[0:17])
print(address[19:32])
print(address[10:100])

print("\n")

print(address[34:-6])
print(address[-8:-6])
print(address[-8:36])
```

string-slicing-I-slicing-by-range.py (Part 2)

```
print("\n")

print(address[5:])
print(address[13:])
print(address[-10:])

print(address[:10])
print(address[0:10])
print(address[:23])
print(address[:-3])

print(address[:])
```

string-slicing-II-slicing-by-steps.py

```
alphabet = "abcdefghijklmnopqrstuvwxyz"

print(alphabet[0:10:2])
print(alphabet[0:26:3])
print(alphabet[:26:3])
print(alphabet[0::3])
print(alphabet[::3])

print(alphabet[4:20:5])
print(alphabet[-20:-8:5])

print(alphabet[::-3])
print(alphabet[::-2])
print(alphabet[::-1])
```

escape-characters.py (Part 1)

```
print("This will \nbegin on a \nnew line")

print("\tOnce upon a time")

print("\"To be or not to be\", said Hamlet")
print('\'To be or not to be\', said Hamlet')
print("Let's print a backslash: \\")

file_name = r"C:\news\travel"
print(file_name)
```

escape-characters.py (Part 2)

```
some_random_number = 5
some_obscure_calculation = 25
some_additional_statistic_fetched_from_somewhere = 10

final = some_random_number + \
        some_obscure_calculation + \
        some_additional_statistic_fetched_from_somewhere

print(some_random_number,
      some_obscure_calculation,
      some_additional_statistic_fetched_from_somewhere)
```

the-in-and-not-in-operators-for-inclusion-and-exclusion.py

```python
announcement = "The winners of the prize are Boris, Andy, and Adam"

print("Boris" in announcement)
print("Steven" in announcement)
print("boris" in announcement)
print(" " in announcement)
print("," in announcement)

print()

print("Boris" not in announcement)
print("Steven" not in announcement)
print("boris" not in announcement)
print(" " not in announcement)
print("," not in announcement)
```

7

Strings: Methods

the-find-and-index-methods.py (Part 1)

```
browser = "Google Chrome"

print(browser.find("C"))
print(browser.find("Ch"))
print(browser.find("o"))
print(browser.find("G"))
print(browser.find("Z"))
print(browser.find("Zxy"))
print(browser.find("c"))
```

the-find-and-index-methods.py (Part 2)

```python
print()

print(browser.find("o"))
print(browser.find("o", 2))
print(browser.find("o", 5))

print("Ch" in browser)

print(browser.index("C"))
# print(browser.index("Z"))
```

the-startswith-and-endswith-methods.py

```python
salutation = "Mr. Kermit the Frog"

print(salutation.startswith("M"))
print(salutation.startswith("Mr"))
print(salutation.startswith("Mr."))
print(salutation.startswith("m"))
print(salutation.startswith("mr."))
print(salutation.startswith("Ker"))
print(salutation.startswith("Mr. Ker"))

print(salutation.endswith("g"))
print(salutation.endswith("og"))
print(salutation.endswith("Frog"))
print(salutation.endswith("frog"))
```

the-count-method.py

```python
word = "queueing"

print(word.count("e"))
print(word.count("u"))
print(word.count("q"))
print(word.count("z"))
print(word.count("Q"))

print(word.count("ue"))
print(word.count("ing"))
print(word.count("u") + word.count("e"))
```

the-capitalize-title-lower-upper-and-swapcase-methods.py

```
story = "once upon a time"

print(story.capitalize())
print(story.title())
print(story.upper())
print("HELLO".lower())
print("AbCdE".swapcase())
print("BENJAMIN FRANKLIN".lower().title())

story = story.title()
print(story)
```

boolean-methods-for-strings.py (Part 1)

```
print("winter".islower())
print("winter 12#$".islower())
print("Winter 12#$".islower())

print("SUMMER".isupper())
print("SUMMER 34%&".isupper())
print("sUMMER 34%&".isupper())

print("The Four Seasons".istitle())
print("The 4 Seasons".istitle())
print("The four Seasons".istitle())
```

boolean-methods-for-strings.py (Part 2)

```
print("area".isalpha())
print("Area 51".isalpha())

print("51".isnumeric())
print("Area 51".isnumeric())

print("Area51".isalnum())
print("Area 51".isalnum())

print(" ".isspace())
print("   ".isspace())
print(" k  ".isspace())
print(" 4  ".isspace())
print(" !  ".isspace())
```

the-lstrip-rstrip-and-strip-methods.py

```python
empty_space = "          content                "
print(empty_space.rstrip())
print(len(empty_space.rstrip()))

print(empty_space.lstrip())
print(len(empty_space.lstrip()))

print(empty_space.strip())
print(len(empty_space.strip()))

website = "www.python.org"
print(website.lstrip("w"))
print(website.rstrip("org"))
print(website.strip("worg."))
```

the-replace-method.py

```
phone_number = "555 555 1234"
print(phone_number.replace(" ", "-"))
print(phone_number.replace("5", "9"))
print(phone_number)

phone_number = phone_number.replace(" ", "-")
```

The-format-method.py (Part 1)

```
# name, adjective, noun
mad_libs = "{} laughed at the {} {}."

print(mad_libs.format("Bobby", "green", "alien"))
print(mad_libs.format("Jennifer", "silly", "tomato"))

mad_libs = "{2} laughed at the {1} {0}."

print(mad_libs.format("Bobby", "green", "alien"))
print(mad_libs.format("Jennifer", "silly", "tomato"))
```

The-format-method.py (Part 2)

```python
mad_libs = "{name} laughed at the {adjective} {noun}."

# print(mad_libs.format(name = "Bobby", adjective = "green", noun = "alien"))
# print(mad_libs.format(name = "Jennifer", adjective = "silly", noun = "tomato"))
# print(mad_libs.format(adjective = "silly", noun = "tomato", name = "Jennifer"))

name = input("Enter a name: ")
adjective = input("Enter an adjective: ")
noun = input("Enter a noun: ")

print(mad_libs.format(name = name, adjective = adjective, noun = noun))
```

formatted-string-literals.py

```python
name = "Bobby"
adjective = "green"
noun = "alien"

mad_libs = f"{name} laughed at the {adjective} {noun}."
print(mad_libs)

mad_libs = F"{name} laughed at the {adjective} {noun}."
print(mad_libs)

print(f"2 + 2 is {2 + 2}")
```

8

Control Flow

REVIEW-the-boolean-data-type-equality-and-inequality.py

```
handsome = True
admin = False
print(2 < 4)
print(7 >= 8)
result = 2 < 4
print(result)

print("xbox" == "xbox")
print("xbox" == "playstation")
print("xbox" == "Xbox")
print(5 == 5)
print(5 == 7)

print(4 != 5)
print(5 != 5)
print("Boris" != "boris")
print("Boris" != "Boris")
print(True != False)
print(True != True)
```

the-if-statement.py (Part 1)

```
if 5 > 3:
   print("Yup, that's true. This will be printed!")
   print("Here's another line! Hooray")

if 6 > 10:
   print("Nope, that is FALSE. That will not be printed!")

if "boris" == "boris":
   print("Great name!")
```

the-if-statement.py (Part 2)

```python
if "dave" == "Dave":
   print("Awesome name")

if "dave" != "Dave":
   print("Haha, got you to print")
   print("Great success!")

if True:
   print("Always true, always prints")

if False:
   print("Never true, not fit to print!")
```

the-bool-function-truthiness-and-falsiness.py (Part 1)

```python
if 3:
   print("Hello")

if -1:
   print("Goodbye")

if 0:
   print("Will this print?")

if "hello":
   print("La la la")

if "":
   print("This will not print")
```

the-bool-function-truthiness-and-falsiness.py (Part 2)

```python
print(bool(1))
print(bool(0))
print(bool(""))
print(bool("Python"))
print(bool(3.14))
print(bool(-1.309320))
print(bool(0.0))
```

the-else-statement.py

```python
if 20 > 15:
   print("That is true!")
else:
   print("That is false!")

value = int(input("Enter a random number: "))

if value > 100000:
   print("I like that you're thinking big!")
else:
   print("That's an OK number, I guess")
```

the-elif-statement.py (Part 1)

```python
def positive_or_negative(number):
    if number > 0:
        return "Positive!"
    elif number < 0:
        return "Negative!"
    else:
        return "Neither! It's zero"

print(positive_or_negative(5))
print(positive_or_negative(-3))
print(positive_or_negative(0))
```

the-elif-statement.py (Part 2)

```
def calculator(operation, a, b):
   if operation == "add":
       return a + b
   elif operation == "subtract":
       return a - b
   elif operation == "multiply":
       return a * b
   elif operation == "divide":
       return a / b
   else:
       return "I don't know what you want me to do!"

print(calculator("add", 3, 4))
print(calculator("subtract", 3, 4))
print(calculator("multiply", 3, 4))
print(calculator("divide", 3, 4))
print(calculator("transmogrify", 3, 4))
print(calculator("", 3, 4))
```

conditional-expressions.py

```python
zip_code = "902101"

# if len(zip_code) == 5:
#     check = "Valid"
# else:
#     check = "Invalid"

check = "Valid" if len(zip_code) == 5 else "Invalid"
print(check)
```

the-and-keyword.py

```python
if 5 < 7 and "rain" == "rain":
   print("Both of those conditions evaluate to True")

if 5 < 7 and "rain" == "fire":
   print("This will not print because at least one of the two conditions is false")

if "rain" == "fire" and 5 < 7:
   print("This will not print because at least one of the two conditions is false")

if "rain" == "fire" and 5 < 3:
   print("This will not print because at least one of the two conditions is false")

value = 95

# if value > 90 and value < 100:
if 90 < value < 100:
   print("This is a shortcut for the win!")
```

the-or-keyword.py

```python
if 5 > 8 or 7 < 11:
    print("At least one of the conditions is True!")

if "cat" == "cat" or "dog" == "donkey":
    print("At least one of the conditions is True!")

if "cat" == "cat" or "dog" == "dog":
    print("At least one of the conditions is True!")

if "apple" == "banana" or "orange" == "pear":
    print("Will this print? Nope!")
```

the-not-keyword.py (Part 1)

```
print(not True)
print(not False)

if "H" in "Hello":
   print("That character exists in the string!")

if "Z" not in "Hello":
   print("That character does not exist in the string!")
```

the-not-keyword.py (Part 2)

```python
value = 10

if value > 100:
    print("This will NOT print!")

if value < 100:
    print("This will print")

if not value > 100:
    print("This will print!")
```

nested-if-statements.py

```python
ingredient1 = "Pizza"
ingredient2 = "Sausage"

if ingredient1 == "Pasta":
    if ingredient2 == "Meatballs":
        print("I recommend making pasta and meatballs")
    else:
        print("I recommend making plain pasta")
else:
    print("I have no recommendations")

if ingredient1 == "Pasta" and ingredient2 == "Meatballs":
    print("I recommend making pasta and meatballs")
elif ingredient1 == "Pasta":
    print("I recommend making plain pasta")
else:
    print("I have no recommendations")
```

the-while-loop.py (Part 1)

```python
count = 0

while count <= 5:
    print(count)
    count += 1

print(count)

count = 0

while count <= 5:
    print(count)
    count += 1
```

the-while-loop.py (Part 2)

```
invalid_number = True

while invalid_number:
   user_value = int(input("Please enter a number above 10: "))
   if user_value > 10:
       print(f"Thanks, that works! {user_value} is a great choice!")
       invalid_number = False
   else:
       print("That doesn't fit! Try again!")
```

a-brief-intro-to-recursion.py

```python
# def count_down_from(number):
#     start = number
#     while start > 0:
#         print(start)
#         start -= 1

def count_down_from(number):
   if number <= 0:
       return

   print(number)
   count_down_from(number - 1)

count_down_from(5)
```

a-brief-intro-to-recursion-II.py

```python
# def reverse(str):
#     start_index = 0
#     last_index = len(str) - 1    # -1
#     reversed_string = ""         # warts

#     while last_index >= start_index:
#         reversed_string += str[last_index]
#         last_index -= 1

#     return reversed_string

def reverse(str):
   if len(str) <= 1:
       return str

   return str[-1] + reverse(str[:-1])

print(reverse("straw")) # warts
```

9

Lists: The Basics

intro-to-lists.py

```
empty = []
empty = list()

sodas = ["Coke", "Pepsi", "Dr. Pepper"]
print(len(sodas))

quarterly_revenues = [15000, 12000, 9000, 20000]
print(len(quarterly_revenues))

stock_prices = [343.26, 89.25]
print(len(stock_prices))

user_settings = [True, False, False, True, False]
print(len(user_settings))
```

the-in-and-not-in-operators-on-a-list.py (Part 1)

```python
print("fast" in "breakfast")
print("fast" in "dinner")

meals = ["breakfast", "lunch", "dinner"]

print("lunch" in meals)
print("dinner" in meals)
print("snack" in meals)
print("Breakfast" in meals)
```

the-in-and-not-in-operators-on-a-list.py (Part 2)

```
test_scores = [99.0, 35.0, 23.5]

print(99.0 in test_scores)
print(99 in test_scores)
print(28 in test_scores)
print(43.7 in test_scores)

print("lunch" not in meals)
print("Breakfast" not in meals)
print(1000 not in test_scores)
print(35 not in test_scores)

if 1000 not in test_scores:
   print("That value is not in there!")
```

select-a-list-element-by-positive-or-negative-index-positions.py

```
print("organic"[5])

web_browsers = ["Chrome", "Firefox", "Safari", "Opera", "Edge"]
print(web_browsers[0])
print(web_browsers[1])
print(web_browsers[4])

# print(web_browsers[10])
print(web_browsers[2][3])

presidents = ["Washington", "Adams", "Jefferson"]
print(presidents[-1])
print(presidents[-2])
print(presidents[-3])

# print(presidents[-20])
```

slice-multiple-elements-from-a-list.py

```python
print("programming"[3:6])

muscles = ["Biceps", "Triceps", "Deltoid", "Sartorius"]

print(muscles[1:3])
print(muscles[1:2])
print(muscles[0:2])
print(muscles[:2])
print(muscles[2:100])
print(muscles[2:])
print(muscles[-4:-2])
print(muscles[-3:])
print(muscles[:-1])
print(muscles[1:-1])
print(muscles[::2])
print(muscles[::-2])
print(muscles[::-1])
```

Lists: Iteration

iteration-with-the-for-loop.py (Part 1)

```python
dinner = "Steak and Potatoes"

# for character in dinner:
#     print(character)

numbers = [2, 3, 5, 7, 10]

for number in numbers:
   print(number * number)
```

iteration-with-the-for-loop.py (Part 2)

```
novelists = ["Fitzgerald", "Hemingway", "Steinbeck"]

for novelist in novelists:
   print(len(novelist))

print(novelist)
print(number)

total = 0

for number in numbers:
   total = total + number

print(total)
```

iteration-with-conditional-logic.py (Part 1)

```python
values = [3, 6, 9, 12, 15, 18, 21, 24]
other_values = [5, 10, 15, 20, 25, 30]

def odds_sum(numbers):
    total = 0
    for number in numbers:
        if number % 2 == 1:
            total += number
    return total

print(odds_sum(values))          # 48
print(odds_sum(other_values))    # 45
```

iteration-with-conditional-logic.py (Part 2)

```python
def greatest_number(numbers):
    greatest = numbers[0]
    for number in numbers:
        if number > greatest:
            greatest = number
    return greatest

print(greatest_number([1, 2, 3])) # 3
print(greatest_number([3, 2, 1])) # 3
print(greatest_number([4, 5, 5])) # 5
print(greatest_number([-3, -2, -1])) # -1
```

iterate-in-reverse-with-the-reversed-function.py

```
the_simpsons = ["Homer", "Marge", "Bart", "Lisa", "Maggie"]

for character in the_simpsons[::-1]:
    print(f"{character} has a total of {len(character)} characters.")

print(reversed(the_simpsons))
print(type(reversed(the_simpsons)))

for character in reversed(the_simpsons):
    print(f"{character} has a total of {len(character)} characters.")
```

the-enumerate-function.py

```python
errands = ["Go to gym", "Grab lunch", "Get promoted at work", "Sleep"]

print(enumerate(errands))

for idx, task in enumerate(errands, 1):
   print(f"{task} is number {idx} on my list of things to do today!")
```

the-range-function.py

```python
for number in range(11):
    print(number)

for number in range(3, 9):
    print(number)

for number in range(10, 101, 8):
    print(number)

for number in range(99, -1, -11):
    print(number)
```

the-break-keyword.py

```python
print(3 in [1, 2, 3, 4, 5, 3])

def contains(values, target):
    found = False

    for value in values:
        print(value)
        if value == target:
            found = True
            break

    return found

print(contains([1, 2, 3, 4, 5], 3))
```

the-continue-keyword.py

```python
def sum_of_positive_numbers(values):
    total = 0

    for value in values:
        if value < 0:
            continue

        total += value

    return total

print(sum_of_positive_numbers([1, 2, -3, 4]))
```

command-line-arguments-with-argv.p

```python
import sys

# print(sys.argv)
# print(type(sys.argv))

word_lengths = 0

for arg in sys.argv[1:]:
    word_lengths += len(arg)

print(f"The total length of all command-line arguments is {word_lengths}")
```

11

Debugging

intro-to-debugging-in-vscode.py

```python
# Define a function that iterates over a list of numbers,
# multiplies each number by one less than its index position
# and returns the total sum of those products

values = [1, 2, 3, 4, 5]

def multiply_element_by_one_less_than_index(numbers):
    total = 0
    for index, number in enumerate(numbers):
        total += number * (index - 1)
    return total

print(multiply_element_by_one_less_than_index(values))
```

working-through-a-problem.py

```python
import sys

# print(sys.argv)
# print(type(sys.argv))

word_lengths = 0

for arg in sys.argv[1:]:
    word_lengths += len(arg)

print(f"The total length of all command-line arguments is {word_lengths}")
```

12

Lists: Mutation

assign-new-value-at-index.py

```python
crayons = ["Macaroni and Cheese", "Maximum Yellow Red", "Jazzberry Jam"]
print(crayons)

crayons[1] = "Cotton Candy"
print(crayons)

crayons[0] = "Blizzard Blue"
print(crayons)

crayons[-1] = "Aquamarine"
print(crayons)

# crayons[3] = "Aztec Gold"
```

assign-new-values-to-list-slice.py

```python
coworkers = ["Michael", "Jim", "Dwight", "Pam", "Creed", "Angela"]

# coworkers[3:5] = ["Oscar", "Ryan"]
# print(coworkers)

# coworkers[3:5] = ["Oscar"]
# print(coworkers)

# coworkers[3:5] = ["Oscar", "Ryan", "Meredith"]
# print(coworkers)

coworkers[-3:-1] = ["Ryan"]
print(coworkers)
```

the-append-method.py

```
countries = ["United States", "Canada", "Australia"]
print(countries)
print(len(countries))

countries.append("Japan")
print(countries)
print(len(countries))

countries.append("France")
print(countries)
print(len(countries))

countries.append("Belgium")
print(countries)
```

building-a-list-up-from-another-list.py (Part 1)

```python
powerball_numbers = [4, 8, 15, 16, 23, 42]

def squares(numbers):
    squares = []
    for number in numbers:
        squares.append(number ** 2)
    return squares

print(squares(powerball_numbers)) # [16, 64, ...]
```

building-a-list-up-from-another-list.py (Part 2)

```
def convert_to_floats(numbers):
    floats = []
    for number in numbers:
        floats.append(float(number))
    return floats

print(convert_to_floats(powerball_numbers))
print(convert_to_floats([10, 67, 23]))
```

building-a-list-up-from-another-list.py (Part 3)

```
def even_or_odd(numbers):
    results = []
    for number in numbers:
        if number % 2 == 0:
            results.append(True)
        else:
            results.append(False)
    return results

print(even_or_odd(powerball_numbers)) # [True, True, False,
True, False, True]
```

the-extend-method.py (Part 1)

```python
mountains = ["Mount Everest", "K2"]
print(mountains)

mountains.extend(["Kangchenjunga", "Lhotse", "Makalu"])
print(mountains)

extra_mountains = ["Cho Oyu", "Dhaulagiri"]
mountains.extend(extra_mountains)
print(mountains)

mountains.extend([])
print(mountains)
```

the-extend-method.py (Part 2)

```python
steaks = ["Tenderloin", "New York Strip"]
more_steaks = ["T-Bone", "Ribeye"]

my_meal = steaks + more_steaks
print(my_meal)
print(steaks)
print(more_steaks)

steaks += more_steaks
print(steaks)
```

the-insert-method.py

```python
plays = ["Hamlet", "Macbeth", "King Lear"]

plays.insert(1, "Julius Caesar")
print(plays)

plays.insert(0, "Romeo & Juliet")
print(plays)

plays.insert(10, "A Midsummer Night's Dream")
print(plays)
```

the-pop-method.py

```python
action_stars = ["Norris", "Seagal", "Van Damme", "Schwarzenegger"]

# last_action_hero = action_stars.pop()
# print(action_stars)
# print(last_action_hero)

# action_stars.pop()
# print(action_stars)

# second_star = action_stars.pop(1)
# print(action_stars)
# print(second_star)

muscles_from_brussels = action_stars.pop(-2)
print(action_stars)
print(muscles_from_brussels)
```

the-del-keyword.py

```python
soups = ["French Onion", "Clam Chowder", "Chicken Noodle",
"Miso", "Wonton"]

# del soups[1]

# del soups[-1]

del soups[1:3]
print(soups)
```

the-remove-method.py

```
nintendo_games = ["Zelda", "Mario", "Donkey Kong", "Zelda"]

nintendo_games.remove("Zelda")
print(nintendo_games)

nintendo_games.remove("Zelda")
print(nintendo_games)

if "Wario" in nintendo_games:
   nintendo_games.remove("Wario")

if "Mario" in nintendo_games:
   nintendo_games.remove("Mario")

print(nintendo_games)
```

the-clear-method.py

```
citrus_fruits = ["Lemon", "Orange", "Lime"]
citrus_fruits.clear()
print(citrus_fruits)
```

the-reverse-method.py

```
vitamins = ["A", "D", "K"]
vitamins.reverse()
print(vitamins)
```

the-sort-method.py

```
temperatures = [40, 28, 52, 66, 35]
temperatures.sort()
temperatures.reverse()
print(temperatures)

coffees = ["Latte", "Espresso", "Macchiato", "Frappucino"]
coffees.sort()
coffees.reverse()
print(coffees)

coffees = ["Latte", "espresso", "macchiato", "Frappucino"]
coffees.sort()
print(coffees)

coffees = ["Latte", "Espresso", "Macchiato", "Frappucino"]
print(sorted(coffees))
print(coffees)
```

13

Lists: Methods

the-count-method.py

```python
car_lot = ["Ford", "Dodge", "Toyota", "Ford", "Toyota", "Chevrolet",
"Ford"]

print(car_lot.count("Dodge"))
print(car_lot.count("Toyota"))
print(car_lot.count("Ferrari"))
print(car_lot.count("dodge"))

hours_of_sleep = [7.3, 7.0, 8.0, 6.5, 7.0, 8.0]

print(hours_of_sleep.count(7.3))
print(hours_of_sleep.count(7.0))
print(hours_of_sleep.count(7))
```

the-index-method.py

```python
pizzas = [
   "Mushroom", "Pepperoni",
   "Sausage", "Barbecue Chicken",
   "Pepperoni", "Sausage"
]

print(pizzas.index("Barbecue Chicken"))
print(pizzas.index("Pepperoni"))
print(pizzas.index("Sausage"))

if "Olives" in pizzas:
   print(pizzas.index("Olives"))

print(pizzas.index("Pepperoni", 2))
print(pizzas.index("Sausage", 3))
print(pizzas.index("Sausage", 2))
```

the-copy-method.py

```
units = ["meter", "kilogram", "second", "ampere", "kelvin", "candela", "mole"]

more_units = units.copy()

# print(units)
# print(more_units)

units.remove("kelvin")
print(units)
print(more_units)

even_more_units = units[:]
print(even_more_units)
```

the-split-method-on-a-string.py

```
users = "Bob, Dave, John, Sue, Randy, Meg"

print(users.split(", "))
print(users.split(", ", 3))

sentence = "I am learning how to code"
words = sentence.split(" ")
print(words)
```

the-join-method-on-a-string.py

```python
address = ["500 Fifth Avenue", "New York", "NY", "10036"]

print(",".join(address))
print(", ".join(address))
print("".join(address))

print("-".join(["555", "123", "4567"]))
print("|".join(["555", "123", "4567"]))
print("***".join(["555", "123", "4567"]))
```

the-zip-function.py

```python
breakfasts = ["Eggs", "Cereal", "Banana"]
lunches = ["Sushi", "Chicken Teriyaki", "Soup"]
dinners = ["Steak", "Meatballs", "Pasta"]

# print(zip(breakfasts, lunches, dinners))
# print(type(zip(breakfasts, lunches, dinners)))
# print(list(zip(breakfasts, lunches, dinners)))

for breakfast, lunch, dinner in zip(breakfasts, lunches, dinners):
   print(f"My meal for today was {breakfast} and {lunch} and {dinner}.")
```

multidimensional-lists.py (Part 1)

```
bubble_tea_flavors = [
   ["Honeydew", "Mango", "Passion Fruit"],
   ["Peach", "Plum", "Strawberry", "Taro"],
   ["Kiwi", "Chocolate"]
]

# print(len(bubble_tea_flavors))
# print(bubble_tea_flavors[0])
# print(bubble_tea_flavors[1])
# print(bubble_tea_flavors[-1])
# print(len(bubble_tea_flavors[1]))
```

multidimensional-lists.py (Part 2)

```python
# print(bubble_tea_flavors[1][2])
# print(bubble_tea_flavors[0][0])
# print(bubble_tea_flavors[2][1])

all_flavors = []

for flavor_pack in bubble_tea_flavors:
    for flavor in flavor_pack:
        all_flavors.append(flavor)

print(all_flavors)
```

list-comprehensions-I-the-basics.py (Part 1)

```
numbers = [3, 4, 5, 6, 7]
# squares = []

# for number in numbers:
#     squares.append(number ** 2)

# print(squares)
# print(number)

squares = [number ** 2 for number in numbers]
print(squares)
```

list-comprehensions-I-the-basics.py (Part 2)

```python
rivers = ["Amazon", "Nile", "Yangtze"]
print([len(river) for river in rivers])

expressions = ["lol", "rofl", "lmao"]
print([expression.upper() for expression in expressions])

decimals = [4.95, 3.28, 1.08]
print([int(decimal) for decimal in decimals])
```

list-comprehensions-II-filtering.py

```python
print(["abcdefghijklmnopqrstuvwxyz".index(char) for char in "donut"])
print([number / 2 for number in range(20)])

donuts = ["Boston Cream", "Jelly", "Vanilla Cream", "Glazed", "Chocolate Cream"]

# creamy_donuts = []

# for donut in donuts:
#     if "Cream" in donut:
#         creamy_donuts.append(donut)

creamy_donuts = [donut for donut in donuts if "Cream" in donut]
print(creamy_donuts)
print([len(donut) for donut in donuts if "Cream" in donut])
print([donut.split(" ")[0] for donut in donuts if "Cream" in donut])
```

14

Built-In Functions

the-help-function.py

```python
# help(len)
# help(print)
# help("len")
# help("print")

# help(str)
# help(int)
# help(list)

# help("Hello".replace)
# help("mystery".swapcase)
help([1].extend)
```

the-map-function.py

```
numbers = [4, 8, 15, 16, 23, 42]
cubes = [number ** 3 for number in numbers]
print(cubes)

def cube(number):
   return number ** 3

print(list(map(cube, numbers)))

animals = ["cat", "bear", "zebra", "donkey", "cheetah"]
print(list(map(len, animals)))
```

the-filter-function.py

```
animals = ["elephant", "horse", "cat", "giraffe", "cheetah", "dog"]
long_words = [animal for animal in animals if len(animal) > 5]
print(long_words)

def is_long_animal(animal):
    return len(animal) > 5

print(list(filter(is_long_animal, animals)))
```

lambda-functions.py

```
metals = ["gold", "silver", "platinum", "palladium"]

print(list(filter(lambda metal: len(metal) > 5, metals)))
print(list(filter(lambda el: len(el) > 5, metals)))
print(list(filter(lambda word: "p" in word, metals)))

print(list(map(lambda word: word.count("l"), metals)))
print(list(map(lambda val: val.replace("s", "$"), metals)))
```

the-all-and-any-functions.py

```
print(all([True]))
print(all([True, True]))
print(all([True, True, False]))
print(all([1, 2, 3]))
print(all([1, 2, 3, 0]))
print(all(["a", "b"]))
print(all(["a", "b", ""]))
print(all([]))

print(any([True, False]))
print(any([False, False]))
print(any([0, 1]))
print(any([0]))
print(any([" ", ""]))
print(any([""]))
print(any([]))
```

the-max-and-min-functions.py

```
print(max([3, 5, 7]))
print(max(3, 5, 7, 9))
print(min([3, 5, 7]))
print(min(3, 5, 7))

print(max(["D", "Z", "K"]))
print(max("D", "Z", "K"))
print(min(["D", "Z", "K"]))
print(min("D", "Z", "K"))
```

the-sum-function.py

```
print(sum([2, 3, 4]))
print(sum([-2, 3, -4]))

print(sum([-1.3, 4.6, 7.34]))
```

the-dir-function.py

```python
# print(dir([]))
print(dir("pasta"))

print(len("pasta"))
print("pasta".__len__())

print("st" in "pasta")
print("pasta".__contains__("st"))

print("pasta" + " and meatballs")
print("pasta".__add__(" and meatballs"th))
```

the-format-function.py

```python
number = 0.123456789

print(format(number, "f"))
print(type(format(number, "f")))

print(format(number, ".2f"))
print(format(number, ".1f"))
print(format(number, ".3f"))

# print(format(0.5, "f"))
print(format(0.5, "%"))
print(format(0.5, ".2%"))

print(format(8123456, ","))
```

15

Tuples

intro-to-tuples.py

```
foods = "Sushi", "Steak", "Guacamole"
foods = ("Sushi", "Steak", "Guacamole")
print(type(foods))

empty = ()
print(type(empty))

# mystery = (1)
# print(type(mystery))
mystery = 1,
print(type(mystery))

mystery = (1, )
print(type(mystery))
print(tuple(["Sushi", "Steak", "Guacamole"]))
print(type(tuple(["Sushi", "Steak", "Guacamole"])))
print(tuple(["abc"]))
```

lists-vs-tuples.py (Part 1)

```python
birthday = (4, 12, 1991)

# print(len(birthday))

# print(birthday[0])
# print(birthday[1])
# print(birthday[2])

# print(birthday[15])

# print(birthday[-1])
# print(birthday[-2])
# print(birthday[-3])
# print(birthday[-4])
```

lists-vs-tuples.py (Part 2)

```python
# birthday[1] = 13

addresses = (
   ['Hudson Street', 'New York', 'NY'],
   ['Franklin Street', 'San Francisco', 'CA']
)

addresses[1][0] = "Polk Street"
# print(addresses)

print(dir(birthday))
```

unpacking-a-tuple-I-the-basics.py (Part 1)

```python
employee = ("Bob", "Johnson", "Manager", 50)

# first_name = employee[0]
# last_name = employee[1]
# position = employee[2]
# age = employee[3]

# first_name, last_name, position, age = employee
# print(first_name, last_name, position, age)

# subject, verb, adjective = ["Python", "is", "fun"]
# print(subject)
# print(verb)
# print(adjective)
```

unpacking-a-tuple-I-the-basics.py (Part 2)

```
# first_name, last_name, title = employee
# first_name, last_name, position, age, salary = employee

a = 5
b = 10

b, a = a, b
print(a)
print(b)
```

unpacking-a-tuple-II-using-*-to-destructure-multiple-elements.py

```
employee = ("Bob", "Johnson", "Manager", 50)
first_name, last_name, *details = employee
print(first_name)
print(last_name)
print(details)

*names, position, age = employee
print(position)
print(age)
print(names)

first_name, *details, age = employee
print(first_name)
print(age)
print(details)

first_name, last_name, position, *details = employee
print(details)
```

variable-number-of-function-arguments-with-*args.py (Part 1)

```python
def accept_stuff(*args):
    print(type(args))
    print(args)

accept_stuff(1)
accept_stuff(1, 3, 5)
accept_stuff(1, 2, 3, 4, 5)
accept_stuff()
```

variable-number-of-function-arguments-with-*args.py (Part 2)

```python
def my_max(*numbers, nonsense):
    print(nonsense)
    greatest = numbers[0]
    for number in numbers:
        if number > greatest:
            greatest = number
    return greatest

print(my_max(1, 2, 3, 4, nonsense = "Shazam"))
print(my_max(1, 3, nonsense = "Hoorah"))
print(my_max(1, 3, 9, 6, 7, 8, -14, nonsense = "Bonanaza"))
```

unpacking-arguments-to-functions.py

```python
def product(a, b):
    return a * b

# print(product(3, 5))

numbers = [3, 5]
numbers = (3, 5)

print(product(*numbers))
```

16

Objects and References

variables-objects-and-garbage-collection.py

```
a = 3
a = 10
a = "hello"
a = [1, 2, 3]

a = [4, 5, 6]
```

shared-references-with-immutable-and-mutable-types.py

```python
a = 3
b = a

a = 5
print(a)
print(b)

a = [1, 2, 3]
b = a

a.append(4)
print(a)
print(b)

b.append(5)
print(a)
print(b)
```

equality-vs-identity.py (Part 1)

```python
students = ["Bob", "Sally", "Sue"]
athletes = students
nerds = ["Bob", "Sally", "Sue"]

print(students == athletes)
print(students == nerds)

print(students is athletes)
print(students is nerds)
```

equality-vs-identity.py (Part 2)

```python
a = 1
b = 1
print(a == 1)
print(a is b)

a = 3.14
b = 3.14
print(a == b)
print(a is b)

a = "hello"
b = "hello"
print(a == b)
print(a is b)
```

shallow-and-deep-copies.py (Part 1)

```python
import copy

# a = [1, 2, 3]

# b = a[:]
# print(a == b)
# print(a is b)

# c = a.copy()
# print(a == c)
# print(a is c)

# d = copy.copy(a)
# print(a == d)
# print(a is d)

numbers = [2, 3, 4]
a = [1, numbers, 5]
```

shallow-and-deep-copies.py (Part 2)

```python
b = a[:]
b = a.copy()
b = copy.copy(a)
b = copy.deepcopy(a)

print(a == b)
print(a is b)
print(a[1] is b[1])

a[1].append(100)
print(b)
print(a)

b[1].append(200)
print(b)
print(a)
```

17

Dictionaries: The Basics

intro-to-dictionaries.py

```python
ice_cream_preferences = {
    "Benjamin": "Chocolate",
    "Sandy": "Vanilla",
    "Marv": "Cookies & Creme",
    "Julia": "Chocolate"
}

print(len(ice_cream_preferences))
```

access-a-dictionary-value-by-key-or-the-get-method.py (Part 1)

```python
flight_prices = {
    "Chicago": 199,
    "San Francisco": 499,
    "Denver": 295
}

print(flight_prices["Chicago"])
print(flight_prices["Denver"])
# print(flight_prices["Seattle"])
# print(flight_prices["chicago"])
```

access-a-dictionary-value-by-key-or-the-get-method.py (Part 2)

```python
gym_membership_packages = {
   29: ["Machines"],
   49: ["Machines", "Vitamin Supplements"],
   79: ["Machines", "Vitamin Supplements", "Sauna"]
}

print(gym_membership_packages[49])
print(gym_membership_packages[79])
# print(gym_membership_packages[100])

print(gym_membership_packages.get(29, ["Basic Dumbbells"]))
print(gym_membership_packages.get(100, ["Basic Dumbbells"]))
print(gym_membership_packages.get(100))
```

the-in-and-not-in-operators-on-a-dictionary.py (Part 1)

```
# print("erm" in "watermelon")
# print("z" in "watermelon")
# print("z" not in "watermelon")

# print(10 in [10, 20, 25])
# print(30 in [10, 20, 25])
# print(30 not in [10, 20, 25])
```

the-in-and-not-in-operators-on-a-dictionary.py (Part 2)

```python
pokemon = {
   "Fire": ["Charmander", "Charmeleon", "Charizard"],
   "Water": ["Squirtle", "Warturtle", "Blastoise"],
   "Grass": ["Bulbasaur", "Venusaur", "Ivysaur"]
}

print("Fire" in pokemon)
print("Grass" in pokemon)
print("Electric" in pokemon)
print("fire" in pokemon)
print("Electric" not in pokemon)
print("fire" not in pokemon)
print("Zombie" not in pokemon)
print("Water" not in pokemon)

if "Zombie" in pokemon:
   print(pokemon["Zombie"])
else:
   print("The category of Zombie does not exist!")
```

add-or-modify-key-value-pair-in-dictionary.py (Part 1)

```
sports_team_rosters = {
    "New England Patriots": ["Tom Brady", "Rob Gronkowski", "Julian Edelman"],
    "New York Giants": ["Eli Manning", "Odell Beckham"]
}

# print(sports_team_rosters["Pittsburgh Steelers"])
sports_team_rosters["Pittsburgh Steelers"] = ["Ben Roethlisberger", "Antonio
Brown"]
# print(sports_team_rosters["Pittsburgh Steelers"])
# print(sports_team_rosters)

sports_team_rosters["New York Giants"] = ["Eli Manning"]
```

add-or-modify-key-value-pair-in-dictionary.py (Part 2)

```
video_game_options = {}
# video_game_options = dict()

video_game_options["subtitles"] = True
video_game_options["difficulty"] = "Medium"
video_game_options["volume"] = 7
print(video_game_options)

video_game_options["difficulty"] = "Hard"
video_game_options["subtitles"] = False
video_game_options["Volume"] = 10
print(video_game_options)
```

add-or-modify-key-value-pair-in-dictionary.py (Part 3)

```python
words = ["danger", "beware", "danger", "beware", "beware"]

def count_words(words):
    counts = {}
    for word in words:
        if word in counts:
            # counts[word] = counts[word] + 1
            counts[word] += 1
        else:
            counts[word] = 1
    return counts

print(count_words(words))
```

the-setdefault-method.py

```python
film_directors = {
   "The Godfather": "Francis Ford Coppola",
   "The Rock": "Michael Bay",
   "Goodfellas": "Martin Scorsese"
}

print(film_directors.get("Goodfellas"))
print(film_directors.get("Bad Boys", "Michael Bay"))
print(film_directors)

# film_directors.setdefault("Bad Boys")
# print(film_directors)

film_directors.setdefault("Bad Boys", "Michael Bay")
print(film_directors)

film_directors.setdefault("Bad Boys", "A good director")
print(film_directors)
```

the-pop-method.py (Part 1)

```python
# years = [1991, 1995, 2000, 2007]
# years.pop(1)
# print(years)

release_dates = {
    "Python": 1991,
    "Ruby": 1995,
    "Java": 1995,
    "Go": 2007
}

# year = release_dates.pop("Java")
# print(release_dates)
# print(year)
```

the-pop-method.py (Part 2)

```python
# release_dates.pop("Go")
# print(release_dates)

# if "Rust" in release_dates:
#     release_dates.pop("Rust")

# new_year = release_dates.pop("Ruby", 2000)
# print(new_year)

del release_dates["Java"]
print(release_dates)

del release_dates["Rust"]
```

the-clear-method.py

```python
websites = {
    "Wikipedia": "http://www.wikipedia.org",
    "Google": "http://www.google.com",
    "Netflix": "http://www.netflix.com"
}

websites.clear()
print(websites)

del websites
# print(websites)
```

the-update-method.py

```python
employee_salaries = {
   "Guido": 100000,
   "James": 500000,
   "Brandon": 900000
}

extra_employee_salaries = {
   "Yukihiro": 1000000,
   "Guido": 333333
}

# employee_salaries.update(extra_employee_salaries)
extra_employee_salaries.update(employee_salaries)

print(employee_salaries)
print(extra_employee_salaries)
```

the-dict-function.py

```python
print(list("abc"))
print(str(9))
print(dict()) # {}

employee_titles = [
   ["Mary", "Senior Manager"],
   ["Brian", "Vice President"],
   ["Julie", "Assistant Vice President"]
]

print(dict(employee_titles))
```

nested-dictionaries.py

```
tv_shows = {
   "The X-Files": {
       "Season 1": {
           "Episodes": ["Scary Monster", "Scary Alien"], "Genre": "Science Fiction",
           "Year": 1993
       },
       "Season 2": { "Episodes": ["Scary Conspiracy"], "Genre": "Horror", "Year": 1994 }
   },
   "Lost": {
       "Season 1": {
           "Episodes": ["What The Heck Is Happening On This Island?"],
           "Genre": "Science Fiction", "Year": 2004
       }
   }
}
print(tv_shows["The X-Files"]["Season 1"]["Episodes"][1])
print(tv_shows["The X-Files"]["Season 2"]["Year"])
print(tv_shows["Lost"]["Season 1"]["Genre"])
```

Dictionaries: Iteration

iterate-over-a-dictionary-with-a-for-loop.py

```
chinese_food = {
   "Sesame Chicken": 9.99,
   "Boneless Spare Ribs": 7.99,
   "Fried Rice": 1.99
}

for food in chinese_food:
   print(f"The food is {food} and its price is {chinese_food[food]}")

pounds_to_kilograms = { 5: 2.26796, 10: 4.53592, 25: 11.3398 }

# 5 pounds is equal to 2.26796 kilograms
for weight_in_pounds in pounds_to_kilograms:
   print(f"{weight_in_pounds} pounds is equal to {pounds_to_kilograms[weight_in_pounds]}
kilograms.")
```

the-items-method.py

```python
college_courses = {
   "History": "Mr. Washington",
   "Math": "Mr. Newton",
   "Science": "Mr. Einstein"
}

for course, professor in college_courses.items():
   print(f"The course {course} is being taught by {professor}.")

for _, professor in college_courses.items():
   print(f"The next professor is {professor}")
```

the-keys-and-values-method.py (Part 1)

```python
cryptocurrency_prices = {
   "Bitcoin": 400000,
   "Ethereum": 7000,
   "Litecoin": 10
}

# print(cryptocurrency_prices.keys())
# print(type(cryptocurrency_prices.keys()))

# print(cryptocurrency_prices.values())
# print(type(cryptocurrency_prices.values()))

for currency in cryptocurrency_prices.keys():
   print(f"The next currency is {currency}")
```

the-keys-and-values-method.py (Part 2)

```
for price in cryptocurrency_prices.values():
   print(f"The next price is {price}")

print("Bitcoin" in cryptocurrency_prices.keys())
print("Ripple" in cryptocurrency_prices.keys())

print(400000 in cryptocurrency_prices.values())
print(5000 in cryptocurrency_prices.values())

print(len(cryptocurrency_prices.keys()))
print(len(cryptocurrency_prices.values()))
print(len(cryptocurrency_prices))
```

the-sorted-function.py

```
numbers = [4, 7, 2, 9]
print(sorted(numbers))
print(numbers)

salaries = { "Executive Assistant": 20, "CEO": 100 }

print(sorted(salaries))
print(salaries)

wheel_count = { "truck": 2, "car": 4, "bus": 8 }

for vehicle, count in sorted(wheel_count.items()):
   print(f"The next vehicle is a {vehicle} and it has {count} wheels.")
```

lists-of-dictionaries.py

```
concert_attendees = [
   { "name": "Taylor", "section": 400, "price paid": 99.99 },
   { "name": "Christina", "section": 200, "price paid": 149.99 },
   { "name": "Jeremy", "section": 100, "price paid": 0.0 }
]

for attendee in concert_attendees:
   for key, value in attendee.items():
       print(f"The {key} is {value}.")
```

keyword-arguments.-kwargs.py (Part 1)

```
def length(word):
    return len(word)

print(length("Hello"))
print(length(word = "Hello"))
# print(length())
# print(length(something = "Hello"))
# print(length(word = "Hello", something = "Goodbye"))

def collect_keyword_arguments(**kwargs):
    print(kwargs)
    print(type(kwargs))

    for key, value in kwargs.items():
        print(f"The key is {key} and the value is {value}")
```

keyword-arguments.-kwargs.py (Part 2)

```
collect_keyword_arguments(a = 2, b = 3, c = 4, d = 5)

def args_and_kwargs(a, b, *args, **kwargs):
    print(f"The total of your regular arguments a and b is {a + b}")
    print(f"The total of your args tuple is {sum(args)}")

    dict_total = 0
    for value in kwargs.values():
        dict_total += value

    print(f"The total of your kwargs dictionary is {dict_total}")

args_and_kwargs(1, 2, 3, 4, 5, 6, x = 8, y = 9, z = 10)
```

unpacking-argument-dictionary.py

```
def height_to_meters(feet, inches):
    total_inches = (feet * 12) + inches
    return total_inches * 0.0254

print(height_to_meters(5, 11))

stats = {
    "feet": 5,
    "inches": 11,
}

print(height_to_meters(**stats))
```

dictionary-comprehensions-I.py

```
languages = ["Python", "JavaScript", "Ruby"]

lengths = {
    language: len(language) for language in languages if "t" in language
}

print(lengths)

word = "supercalifragilisticexpialidocious"

letter_counts = {
    letter: word.count(letter) for letter in word if letter > "j"
}
print(letter_counts)
```

dictionary-comprehensions-II.py

```
capitals = {
   "New York": "Albany",
   "California": "Sacramento",
   "Texas": "Austin"
}

inverted = {
    capital: state for state, capital in capitals.items()
    if len(state) != len(capital)
}

print(inverted)
```

19

Sets

intro-to-sets.py (Part 1)

```
stocks = { "MSFT", "FB", "IBM", "FB" }
print(stocks)

prices = { 1, 2, 3, 4, 5, 3, 4, 2 }
print(prices)

lottery_numbers = { (1, 2, 3), (4, 5, 6), (1, 2, 3) }
print(lottery_numbers)

print(len(stocks))
print(len(prices))
print(len(lottery_numbers))
```

intro-to-sets.py (Part 2)

```
print('MSFT' not in stocks)
print('IBM' not in stocks)
print('GOOG' not in stocks)

# for number in prices:
#     print(number)

for numbers in lottery_numbers:
    for number in numbers:
        print(number)


squares = { number ** 2 for number in [-5, -4, -3, 3, 4, 5] }
print(squares)
print(len(squares))
```

the-set-function.py

```python
print(set([1, 2, 3]))
print(set([1, 2, 3, 3, 2, 1]))

print(set((1, 2)))
print(set((1, 2, 1, 2, 1)))

print(set("abc"))
print(set("aabbcc"))

print(set({ "key": "value" }))

philosophers = ["Plato", "Socrates", "Aristotle", "Pythagoras", "Socrates", "Plato"]
philosophers_set = set(philosophers)
philosophers = list(philosophers_set)
print(philosophers)
```

the-add-and-update-methods.py

```python
disney_characters = { "Mickey Mouse", "Minnie Mouse", "Elsa" }

disney_characters.add("Ariel")
print(disney_characters)

disney_characters.add("Elsa")
print(disney_characters)

disney_characters.update(["Donald Duck", "Goofy"])
print(disney_characters)

disney_characters.update(("Simba", "Pluto", "Mickey Mouse"))
print(disney_characters)
```

the-remove-and-discard-methods.py

```python
agents = { "Mulder", "Scully", "Doggett", "Reyes" }

# agents.remove("Doggett")
# print(agents)

# agents.remove("Skinner")

agents.discard("Doggett")
print(agents)

agents.discard("Skinner")
print(agents)
```

the-intersection-method.py

```
candy_bars = { "Milky Way", "Snickers", "100 Grand" }
sweet_things = { "Sour Patch Kids", "Reeses Pieces", "Snickers" }

print(candy_bars.intersection(sweet_things))
print(candy_bars & sweet_things)

values = { 3.0, 4.0, 5.0 }
more_values = { 3, 4, 5, 6 }
print(values.intersection(more_values))
print(more_values.intersection(values))
print(values & more_values)
print(more_values & values)
```

the-union-method.py

```
candy_bars = { "Milky Way", "Snickers", "100 Grand" }
sweet_things = { "Sour Patch Kids", "Reeses Pieces", "Snickers" }

print(candy_bars.union(sweet_things))
print(sweet_things.union(candy_bars))

print(candy_bars | sweet_things)
print(sweet_things | candy_bars)
```

the-difference-method.py

```
candy_bars = { "Milky Way", "Snickers", "100 Grand" }
sweet_things = { "Sour Patch Kids", "Reeses Pieces", "Snickers" }

print(candy_bars.difference(sweet_things))
print(candy_bars - sweet_things)

print(sweet_things.difference(candy_bars))
print(sweet_things - candy_bars)
```

the-symmetric-difference-method.py

```
candy_bars = { "Milky Way", "Snickers", "100 Grand" }
sweet_things = { "Sour Patch Kids", "Reeses Pieces", "Snickers" }

print(candy_bars.symmetric_difference(sweet_things))
print(candy_bars ^ sweet_things)

print(sweet_things.symmetric_difference(candy_bars))
print(sweet_things ^ candy_bars)
```

the-is-subset-and-issuperset-method.

```python
a = { 1, 2, 4 }
b = { 1, 2, 3, 4, 5 }

print(a.issubset(b))
print(a < b)
print(a <= b)
print(b.issubset(a))

print(b.issuperset(a))
print(b > a)
print(b >= a)
print(a.issuperset(b))
```

the-frozenset-object.py

```python
mr_freeze = frozenset([1, 2, 3, 2])
print(mr_freeze)

# mr_freeze.add(4)
# regular_set = { 1, 2, 3 }
# print({ regular_set: "Some value" })

print({ mr_freeze: "Some value" })
```

a-review-of-truthiness-and-falsiness.py (Part 1)

```python
empty_list = []
stuffy_list = [1, 2, 3]

if empty_list:
    print("Empty list has items")

if stuffy_list:
    print("Stuffy list has items")

# if len(stuffy_list) > 0:
#     print("Stuffy list has items")
```

a-review-of-truthiness-and-falsiness.py (Part 2)

```
empty_dict = {}
stuffy_dict = { "a": 5, "b": 10 }

if empty_dict:
   print("Empty dict has key-value pairs")

if stuffy_dict:
   print("Stuffy dict has key-value pairs")

empty_set = set()
stuffy_set = (1, 2, 3)

if empty_set:
   print("Empty set has elements")

if stuffy_set:
   print("Stuffy set has elements")
```

Modules

calculator.py

```python
creator = "Boris"
PI = 3.14159

def add(a, b):
   return a + b

def subtract(a, b):
   return a - b

def area(radius):
   return PI * radius * radius

_year = 2020
```

my_program.py

```python
import calculator

print(calculator.creator)
print(calculator.PI)
print(calculator.add(3, 5))
```

standard_library.py

```python
import string
import math
import this

# print(string.ascii_letters)
# print(string.ascii_lowercase)
# print(string.ascii_uppercase)
# print(string.digits)
# print(string.capwords("hello there"))

# print(math.ceil(4.5))
# print(math.floor(4.8))
# print(math.sqrt(9))
# print(math.sqrt(32))
# print(math.pi)
```

playground.py

```python
import math
import calculator

# import math, calculator
# print(math.__name__)
# print(calculator.__name__)
# print(__name__)

print(calculator.area(5))
```

aliases.py

```python
import calculator as calc
import datetime as dt

print(calc.add(3, 5))
print(dt.datetime(2020, 4, 12))
```

import-specific-attributes.py

```python
from calculator import creator, add, subtract
from math import sqrt
# from some_other_module import creator

print(creator)
print(add(2, 3))
print(subtract(10, 5))
print(sqrt(49))
```

import-all-attributes.py

```python
from calculator import *

print(creator)
print(add(3, 5))
# print(_year)
```

21

Reading From and Writing to Files

read-from-a-file.py

```python
# with open("cupcakes.txt", "r") as cupcakes_file:
#     print("The file has been opened!")
#     content = cupcakes_file.read()
#     print(content)

# print("The file has been closed. We are outside the context block!")

filename = input("What file would you like to open? ")
with open(filename, "r") as file_object:
   print(file_object.read())
```

read-file-line-by-line.py

```
with open("cupcakes.txt") as file_object:
    for line in file_object:
        print(line.strip())
```

write-to-a-file.py

```python
file_name = "my_first_file.txt"

with open(file_name, "w") as file_object:
   file_object.write("Hello file!\n")
   file_object.write("You're my favorite file!")
```

append-to-a-file.py

```
with open("my_first_file.txt", "a") as file_object:
    file_object.write("\nThird line is the best line!")
```

Decorators

higher-order-functions-I.py

```python
def one():
    return 1

print(type(one))

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def calculate(func, a, b):
    return func(a, b)

print(calculate(add, 3, 5))
print(calculate(subtract, 10, 4))
```

nested-functions.py

```python
def convert_gallons_to_cups(gallons):
   def gallons_to_quarts(gallons):
       print(f"Converting {gallons} gallons to quarts!")
       return gallons * 4

   def quarts_to_pints(quarts):
       print(f"Converting {quarts} quarts to pints")
       return quarts * 2

   def pints_to_cups(pints):
       print(f"Converting {pints} pints to cups")
       return pints * 2

   quarts = gallons_to_quarts(gallons)
   pints = quarts_to_pints(quarts)
   cups = pints_to_cups(pints)
   return cups

print(convert_gallons_to_cups(1))
print(convert_gallons_to_cups(3))
# print(pints_to_cups(3))
```

higher-order-functions-II.py (Part 1)

```
def calculator(operation):
   def add(a, b):
       return a + b

   def subtract(a, b):
       return a - b

   if operation == "add":
       return add
   elif operation == "subtract":
       return subtract

print(calculator("add")(10, 4))
print(calculator("subtract")(7, 7))
```

higher-order-functions-II.py (Part 2)

```python
def square(num):
   return num ** 2

def cube(num):
   return num ** 3

def times10(num):
   return num * 10

operations = [square, cube, times10]

for func in operations:
   print(func(5))
```

scope-I-global-vs-local-variables.py

```
age = 28

def fancy_func():
   age = 100
   print(age)

fancy_func()
print(age)
TAX_RATE = 0.08

def calculate_tax(price):
   return round(price * TAX_RATE, 2)

def calculate_tip(price):
   return round(price * (TAX_RATE * 3), 2)

print(calculate_tax(10))
print(calculate_tip(10))
```

scope-II-the-legb-rule.py

```python
def outer():
    # Enclosing function scope

    def inner():
        # Local scope
        return len

    return inner()

print(outer()("python"))
```

scope-III-closures.py

```python
def outer():
    candy = "Snickers"

    def inner():
        return candy

    return inner

the_func = outer()
print(the_func())
```

the-global-keyword.py

```python
# x = 10

def change_stuff():
    global x
    x = 15

# print(x)
change_stuff()
print(x)
```

the-nonlocal-keyword.py

```python
def outer():
   bubble_tea_flavor = "Black"

   def inner():
       nonlocal bubble_tea_flavor
       bubble_tea_flavor = "Taro"

   inner()

   return bubble_tea_flavor

print(outer())
```

intro-to-decorators.py

```python
def be_nice(fn):
    def inner():
        print("Nice to meet you! I'm honored to execute your function for you!")
        fn()
        print("It was my pleasure executing your function! Have a nice day!")

    return inner

@be_nice
def complex_business_logic():
    print("Something complex!")

@be_nice
def another_fancy_function():
    print("Goo goo gaga")

# complex_business_logic()
another_fancy_function()
```

arguments-with-decorator-functions.py

```python
def be_nice(fn):
    def inner(*args, **kwargs):
        print("Nice to meet you! I'm honored to execute your function for you!")
        print(args)
        print(kwargs)
        fn(*args, **kwargs)
        print("It was my pleasure executing your function! Have a nice day!")

    return inner

@be_nice
def complex_business_logic(stakeholder, position):
    print(f"Something complex for our {position} {stakeholder}!")

# complex_business_logic("Boris", "CEO")
complex_business_logic("Boris", position = "CEO")
```

returned-values-from-decorated-functions.py

```python
def be_nice(fn):
    def inner(*args, **kwargs):
        print("Nice to meet you! I'm honored to execute your function for you!")
        result = fn(*args, **kwargs)
        print("It was my pleasure executing your function! Have a nice day!")
        return result

    return inner

@be_nice
def complex_business_sum(a, b):
    return a + b

print(complex_business_sum(a = 3, b = 5))
```

the-functools.wraps-decorator.py

```python
import functools

def be_nice(fn):
 @functools.wraps(fn)
 def inner(*args, **kwargs):
   print("Nice to meet you! I'm honored to run your function for you!")
   result = fn(*args, **kwargs)
   print("It was my pleasure executing your function! Have a nice day!")
   return result

 return inner

@be_nice
def complex_business_sum(a, b):
 "Adds two numbers together"
 return a + b

help(complex_business_sum)
```

Classes: The Basics

class-definition-and-instantiation.py

```python
class Person():
    pass

class DatabaseConnection():
    pass

boris = Person()
sally = Person()

print(boris)
print(sally)

dc = DatabaseConnection()
print(dc)
```

the__init__method.py

```python
class Guitar():
   def __init__(self):
       print(f"A new guitar is being created! This object is {self}")

acoustic = Guitar()
print(acoustic)

electric = Guitar()
print(electric)
```

adding-attributes-to-objects.py

```python
class Guitar():
   def __init__(self):
       print(f"A new guitar is being created! This object is {self}")

acoustic = Guitar()
electric = Guitar()

acoustic.wood = "Mahogany"
acoustic.strings = 6
acoustic.year = 1990

electric.nickname = "Sound Viking 3000"

print(acoustic.wood)
print(electric.nickname)

# print(electric.year)
# print(acoustic.nickname)
```

setting-object-attributes-in-the__init__method.py

```
class Guitar():
    def __init__(self, wood):
        self.wood = wood

acoustic = Guitar("Alder")
electric = Guitar("Mahogany")
print(acoustic.wood)
print(electric.wood)

baritone = Guitar("Alder")

print(baritone.wood)
print(acoustic)
print(baritone)

a = [1, 2, 3]
b = [1, 2, 3]
```

default-values-for-attributes.py

```
class Book():
   def __init__(self, title, author, price = 14.99):
       self.title = title
       self.author = author
       self.price = price

animal_farm = Book("Animal Farm", "George Orwell", 19.99)
gatsby = Book("The Great Gatsby", "F. Scott Fitzgerald")

print(animal_farm.price)
print(gatsby.price)

atlas = Book(title = "Atlas Shrugged", author = "Ayn Rand")
jude = Book(author = "Thomas Hardy", price = 24.99, title = "Jude the Obscure")

print(jude.title)
```

Classes: Attributes and Methods

instance-methods.py (Part 1)

```python
class Pokemon():
   def __init__(self, name, specialty, health = 100):
       self.name = name
       self.specialty = specialty
       self.health = health

   def roar(self):
       print("Raaaaarr!")

   def describe(self):
       print(f"I am {self.name}. I am a {self.specialty} Pokemon!")

   def take_damage(self, amount):
       self.health -= amount
```

instance-methods.py (Part 2)

```
squirtle = Pokemon("Squirtle", "Water")
charmander = Pokemon(name = "Charmander", specialty = "Fire", health = 110)
squirtle.roar()
charmander.roar()
squirtle.describe()
charmander.describe()
print(squirtle.health)
squirtle.take_damage(20)
print(squirtle.health)

squirtle.health = 60
print(squirtle.health)

print(charmander.health)
```

protected-attributes-and-methods.py

```python
class SmartPhone():
   def __init__(self):
       self._company = "Apple"
       self._firmware = 10.0

   def get_os_version(self):
       return self._firmware

   def update_firmware(self):
       print("Reaching out to the server for the next version")
       self._firmware += 1

iphone = SmartPhone()
print(iphone._company)
print(iphone._firmware)
print(iphone.update_firmware())
print(iphone._firmware)
# iphone._firmware = []
```

define-properties-with-property-method.py

```
class Height():
   def __init__(self, feet):
       self._inches = feet * 12

   def _get_feet(self):
       return self._inches / 12

   def _set_feet(self, feet):
       if feet >= 0:
           self._inches = feet * 12

   feet = property(_get_feet, _set_feet)

h = Height(5)
print(h.feet)
h.feet = 6
print(h.feet)
h.feet = -10
print(h.feet)
```

define-properties-with-decorators.py

```python
class Currency():
   def __init__(self, dollars):
       self._cents = dollars * 100

   @property
   def dollars(self):
       return self._cents / 100

   @dollars.setter
   def dollars(self, dollars):
       if dollars >= 0:
           self._cents = dollars * 100

bank_account = Currency(50000)
print(bank_account.dollars)
bank_account.dollars = 100000
print(bank_account.dollars)
bank_account.dollars = -20000
print(bank_account.dollars)
```

the-getattr-and-setattr-functions.py

```
stats = {
    "name": "BBQ Chicken",
    "price": 19.99,
    "size": "Extra Large",
    "ingredients": ["Chicken", "Onions", "BBQ Sauce"]
}

class Pizza():
    def __init__(self, stats):
        for key, value in stats.items():
            setattr(self, key, value)

bbq = Pizza(stats)
print(bbq.size)
print(bbq.ingredients)

for attr in ["price", "name", "diameter", "discounted"]:
    print(getattr(bbq, attr, "Unknown"))
```

the-hasattr-and-deleteattr-functions.py

```
stats = {
   "name": "BBQ Chicken", "price": 19.99,
   "size": "Extra Large", "ingredients": ["Chicken", "Onions", "BBQ Sauce"]
}

class Pizza():
   def __init__(self, stats):
       for key, value in stats.items():
           setattr(self, key, value)

bbq = Pizza(stats)
stats_to_delete = ["size", "diameter", "spiciness", "ingredients"]
print(bbq.size)

for stat in stats_to_delete:
   if hasattr(bbq, stat):
       delattr(bbq, stat)
# print(bbq.size)
```

class-methods.py (Part 1)

```python
class SushiPlatter():
    def __init__(self, salmon, tuna, shrimp, squid):
        self.salmon = salmon
        self.tuna = tuna
        self.shrimp = shrimp
        self.squid = squid

    @classmethod
    def lunch_special_A(cls):
        return cls(salmon = 2, tuna = 2, shrimp = 2, squid = 0)

    @classmethod
    def tuna_lover(cls):
        return cls(salmon = 0, tuna = 10, shrimp = 0, squid = 1)
```

class-methods.py (Part 2)

```
boris = SushiPlatter(salmon = 8, tuna = 4, shrimp = 5, squid = 10)
print(boris.salmon)

lunch_eater = SushiPlatter.lunch_special_A()
print(lunch_eater.salmon)
print(lunch_eater.squid)

tuna_fan = SushiPlatter.tuna_lover()
print(tuna_fan.tuna)
```

class-attributes.py

```python
class Counter():
   count = 0

   def __init__(self):
       Counter.count += 1

   @classmethod
   def create_two(cls):
       two_counters = [cls(), cls()]
       print(f"New Number of Counter objects created: {cls.count}")
       return two_counters

print(Counter.count)
c1 = Counter()
print(Counter.count)
c2, c3 = Counter.create_two()
print(Counter.count)
print(c1.count)
print(c2.count)
print(c3.count)
```

attribute-lookup-order.py

```python
class Counter():
    count = 0

    def __init__(self):
        Counter.count += 1

    @classmethod
    def create_two(cls):
        two_counters = [cls(), cls()]
        print(f"New Number of Counter objects created: {cls.count}")
```

static-methods.py

```
class WeatherForecast():
    def __init__(self, temperatures):
        self.temperatures = temperatures

    @staticmethod
    def convert_from_fahrenheit_to_celsius(fahr):
        calculation = (5/9) * (fahr - 32)
        return round(calculation, 2)

    def in_celsius(self):
        return [self.convert_from_fahrenheit_to_celsius(temp) for temp in self.temperatures]

wf = WeatherForecast([100, 90, 80, 70, 60])
print(wf.in_celsius())
print(WeatherForecast.convert_from_fahrenheit_to_celsius(100))
```

Classes: Magic Methods

intro-to-magic-methods.py

```
print(3.3 + 4.4)
print(3.3.__add__(4.4))

print(len([1, 2, 3]))
print([1, 2, 3].__len__())

print("h" in "hello")
print("hello".__contains__("h"))

print(["a", "b", "c"][2])
print(["a", "b", "c"].__getitem__(2))
```

string-representation-with-the__str__-and-__repr__methods.py

```python
class Card():
   def __init__(self, rank, suit):
       self._rank = rank
       self._suit = suit

   def __str__(self):
       return f"{self._rank} of {self._suit}"

   def __repr__(self):
       return f'Card("{self._rank}", "{self._suit}")'

c = Card("Ace", "Spades")
print(c)
print(str(c))
print(repr(c))
```

sushi.py

```python
"""
A module related to the joy of sushi.
No fishy code found here!
"""

def fish():
   """
   Determines if fish is a good meal choice.
   Always returns True, because it always is.
   """
   return True

class Salmon():
   """
   Blueprint for a Salmon object
   """
   def __init__(self):
       self.tastiness = 10

   def bake(self):
       """
       Bake the fish in an oven.
       """
       self.tastiness += 1
```

docstrings.py

```python
import sushi
import math

# print(sushi.__doc__)
# print(sushi.fish.__doc__)
# print(sushi.Salmon.__doc__)
# print(sushi.Salmon.bake.__doc__)
# print(math.__doc__)
# print(math.sqrt.__doc__)

help(sushi)
```

truthiness-with-the-__bool__-method.py

```
class Emotion():
   def __init__(self, positivity, negativity):
       self.positivity = positivity
       self.negativity = negativity

   def __bool__(self):
       return self.positivity > self.negativity

my_emotional_state = Emotion(positivity = 50, negativity = 75)

if my_emotional_state:
   print("This will NOT print because I have more negativity than positivity.")

my_emotional_state.positivity = 100

if my_emotional_state:
   print("This WILL print because I have more positivity than negativity")
```

namedtuple.py

```python
import collections

Book = collections.namedtuple("Book", ["title", "author"])
# collections.namedtuple("Book", "title author")

animal_farm = Book("Animal Farm", "George Orwell")
gatsby = Book(title = "The Great Gatsby", author = "F. Scott Fitzgerald")

print(animal_farm[0])
print(gatsby[1])
print(animal_farm.title)
print(gatsby.author)
```

length-with-the-__len__-method.py

```python
import collections

Book = collections.namedtuple("Book", ["title", "author"])
animal_farm = Book("Animal Farm", "George Orwell")
gatsby = Book(title = "The Great Gatsby", author = "F. Scott Fitzgerald")

# word = "dynasty"
# print(len(word))
# print(word.__len__())

class Library():
   def __init__(self, *books):
       self.books = books
       self.librarians = []

   def __len__(self):
       return len(self.books)

l1 = Library(animal_farm)
l2 = Library(animal_farm, gatsby)
print(len(l1))
print(len(l2))
```

indexing-with-the-__getitem__-and-__setitem__methods.py (Part 1)

```
# pillows = {
#     "soft": 79.99,
#     "hard": 99.99
# }

# print(pillows["soft"])
# print(pillows.__getitem__("soft"))

class CrayonBox():
   def __init__(self):
       self.crayons = []

   def add(self, crayon):
       self.crayons.append(crayon)

   def __getitem__(self, index):
       return self.crayons[index]

   def __setitem__(self, index, value):
       self.crayons[index] = value
```

indexing-with-the-__getitem__-and-__setitem__methods.py (Part 2)

```
cb = CrayonBox()
cb.add("Blue")
cb.add("Red")

print(cb[0])
print(cb[1])

cb[0] = "Yellow"

print(cb[0])

for crayon in cb:
   print(crayon)
```

the-__del__-method.py

```python
import time

class Garbage():
    def __del__(self):
        print("This is my last breath!")

g = Garbage()

g = [1, 2, 3]

time.sleep(5)

print("Program done!")
```

Classes: Inheritance

define-a-subclass.py

```python
class Store():
    def __init__(self):
        self.owner = "Boris"

    def exclaim(self):
        return "Lots of stuff to buy, come inside!"

class CoffeeShop(Store):
    pass

starbucks = CoffeeShop()

print(starbucks.owner)
print(starbucks.exclaim())
```

new-methods-on-subclasses.py (Part 1)

```
class Employee():
   def do_work(self):
       print("I'm working!")

class Manager(Employee):
   def waste_time(self):
       print("Wow, this YouTube video looks fun!")

class Director(Manager):
   def fire_employee(self):
       print("You're fired!")
```

new-methods-on-subclasses.py (Part 2)

```
e = Employee()
m = Manager()
d = Director()

e.do_work()
# e.waste_time()

m.do_work()
m.waste_time()
# m.fire_employee()

d.do_work()
d.waste_time()
d.fire_employee()
```

override-an-inherited-method-on-a-subclass.py (Part 1)

```python
class Teacher():
   def teach_class(self):
       print("Teaching stuff...")

   def grab_lunch(self):
       print("Yum yum yum!")

   def grade_tests(self):
       print("F! F! F!")

class CollegeProfessor(Teacher):
   def publish_book(self):
       print("Hooray, I'm an author")

   def grade_tests(self):
       print("A! A! A!")
```

override-an-inherited-method-on-a-subclass.py (Part 2)

```
teacher = Teacher()
professor = CollegeProfessor()

teacher.teach_class()
teacher.grab_lunch()
teacher.grade_tests()

professor.publish_book()
professor.grab_lunch()
professor.teach_class()
professor.grade_tests()
```

the-super-function.py

```python
class Animal():
    def __init__(self, name):
        self.name = name

    def eat(self, food):
        return f"{self.name} is enjoying the {food}"

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

watson = Dog("Watson", "Golden Retriever")
print(watson.name)
print(watson.breed)
```

polymorphism-l.py

```python
class Person():
   def __init__(self, name, height):
       self.name = name
       self.height = height

   def __len__(self):
       return self.height

values = [
   "Boris",
   [1, 2, 3],
   (4, 5, 6, 7),
   { "a": 1, "b": 2},
   Person(name = "Boris", height = 71)
]

for value in values:
   print(len(value))
```

polymorphism-II.py (Part 1)

```python
import random

class Player():
   def __init__(self, games_played, victories):
       self.games_played = games_played
       self.victories = victories

   @property
   def win_ratio(self):
       return self.victories / self.games_played

class HumanPlayer(Player):
   def make_move(self):
       print("Let player make the decision!")

class ComputerPlayer(Player):
   def make_move(self):
       print("Run advanced algorithm to calculate best move!")
```

polymorphism-II.py (Part 2)

```
hp = HumanPlayer(games_played = 30, victories = 15)
cp = ComputerPlayer(games_played = 1000, victories = 999)
print(hp.win_ratio)
print(cp.win_ratio)

game_players = [hp, cp]
starting_player = random.choice(game_players)
starting_player.make_move()
```

name-mangling-for-privacy.py (Part 1)

```
class Nonsense():
   def __init__(self):
       self.__some_attribute = "Hello"

   def __some_method(self):
       print("This is coming from some_method!")

class SpecialNonsense(Nonsense):
   pass

n = Nonsense()
sn = SpecialNonsense()
```

name-mangling-for-privacy.py (Part 2)

```python
# print(n.__some_attribute)
# print(n.some_attribute)
# print(sn.__some_attribute)
# print(sn.some_attribute)
# n.__some_method()
# sn.__some_method()

print(n._Nonsense__some_attribute)
print(sn._Nonsense__some_attribute)
n._Nonsense__some_method()
sn._Nonsense__some_method()
```

multiple-inheritance-I.py

```python
class FrozenFood():
   def thaw(self, minutes):
       print(f"Thawing for {minutes} minutes")

   def store(self):
       print("Putting in the freezer!")

class Dessert():
   def add_weight(self):
       print("Putting on the pounds!")

   def store(self):
       print("Putting in the refrigerator!")

class IceCream(Dessert, FrozenFood):
   pass

ic = IceCream()
ic.add_weight()
ic.thaw(5)
ic.store()
print(IceCream.mro())
```

multiple-inheritance-II.py

```python
class Restaurant():
   def make_reservation(self, party_size):
       print(f"Booked a table for {party_size}")

class Steakhouse(Restaurant):
   pass

class Bar():
   def make_reservation(self, party_size):
       print(f"Booked a lounge for {party_size}")

class BarAndGrill(Steakhouse, Bar):
   pass

bag = BarAndGrill()
bag.make_reservation(2)
print(BarAndGrill.mro())
```

multiple-inheritance-III.py

```
class FilmMaker():
   def give_interview(self):
       print("I love making movies!")

class Director(FilmMaker):
   pass

class Screenwriter(FilmMaker):
   def give_interview(self):
       print("I love writing scripts!")

class JackOfAllTrades(Screenwriter, Director):
   pass

stallone = JackOfAllTrades()
stallone.give_interview()
print(JackOfAllTrades.mro())
```

the-isinstance-and-issubclass-functions.py (Part 1)

```python
print(type({ "a": 1}))

print(isinstance(1, int))
print(isinstance({ "a": 1}, dict))
print(isinstance([], list))
print(isinstance([], int))

print(isinstance(1, object))
print(isinstance(3.4, object))
print(isinstance(str, object))
print(isinstance(max, object))

print(isinstance([], (list, dict, int)))
```

the-isinstance-and-issubclass-functions.py (Part 2)

```python
class Person():
   pass

class Superhero(Person):
   pass

arnold = Person()
boris = Superhero()

print(isinstance(boris, Superhero))
print(isinstance(boris, Person))
print(isinstance(arnold, Person))
print(isinstance(arnold, Superhero))

print(issubclass(Superhero, Person))
print(issubclass(Person, Superhero))
print(issubclass(Superhero, object))
print(issubclass(Person, object))
```

composition.py (Part 1)

```
class Paper():
   def __init__(self, text, case):
       self.text = text
       self.case = case

class Briefcase():
   def __init__(self, price):
       self.price = price
       self.papers = []

   def add_paper(self, paper):
       self.papers.append(paper)

   def view_notes(self):
       return [paper.text for paper in self.papers]
```

composition.py (Part 2)

```
class Lawyer():
   def __init__(self, name, briefcase):
       self.name = name
       self.briefcase = briefcase

   def write_note(self, text, case):
       paper = Paper(text, case)
       self.briefcase.add_paper(paper)

   def view_notes(self):
       print(self.briefcase.view_notes())

cheap_briefcase = Briefcase(price = 19.99)
vinny = Lawyer(name = "Vincent", briefcase = cheap_briefcase)

vinny.write_note("My client is innocent!", "AS-2ZK1")
vinny.write_note("There is no evidence of a crime!", "AS-2ZK1")
vinny.view_notes()
```

27

Exception Handling

the-try-except-block.py

```
def divide_five_by_number(n):
    try:
        return 5 / n
    except:
        pass

print(divide_five_by_number(0))
print(divide_five_by_number(10))
print(divide_five_by_number("Nonsense"))
```

catching-one-or-more-specific-exceptions.py

```python
def divide_5_by_number(n):
    try:
        calculation = 5 / n
    except (ZeroDivisionError, TypeError) as e:
        return f"Something went wrong. The error was {e}"
    return calculation

print(divide_5_by_number(10))
print(divide_5_by_number(0))
print(divide_5_by_number("nonsense"))
```

the-raise-keyword.py

```
def add_positive_numbers(a, b):
   try:
       if a <= 0 or b <= 0:
           raise ValueError("One or both of the values is invalid. Both numbers must be positive!")

       return a + b
   except ValueError as e:
       return f"Caught the ValueError: {e}"

print(add_positive_numbers(10, 5))
print(add_positive_numbers(-2, 3))
print(add_positive_numbers(5, -8))
```

user-defined-exceptions.py

```
class NegativeNumbersError(Exception):
    """One or more inputs are negative"""
    pass

def add_positive_numbers(a, b):
    try:
        if a <= 0 or b <= 0:
            raise NegativeNumbersError
    except NegativeNumbersError:
        return "Shame on you, not valid!"

print(add_positive_numbers(-5, -2))
```

exception-inheritance-hierarchies.py

```
class Mistake(Exception):
   pass

class StupidMistake(Mistake):
   pass

class SillyMistake(Mistake):
   pass

try:
   raise StupidMistake("Extra stupid mistake")
except StupidMistake as e:
   print(f"Caught the error: {e}")

try:
   raise StupidMistake("Extra stupid mistake")
except Mistake as e:
   print(f"Caught the error: {e}")

try:
   raise SillyMistake("Super silly mistake")
except Mistake as e:
   print(f"Caught the error: {e}")
```

the-else-and-finally-blocks.py

```python
x = 10

try:
    print(x + 5)
except NameError:
    print("Some variable is not defined!")
else:
    print("This will print if there is no error in the try.")
finally:
    print("This will print with or without exception")
    print("Closing file...")
```

28

Dates and Time

the-date-object.py

```python
# import datetime
from datetime import date

birthday = date(1991, 4, 12)
print(birthday)
print(type(birthday))
moon_landing = date(year = 1969, month = 7, day = 20)
print(moon_landing)

# date(2025, 15, 10)
# date(2020, 1, 35)

print(birthday.year)
print(birthday.month)
print(birthday.day)
# birthday.year = 2000

today = date.today()
print(today)
print(type(today))
```

the-time-object.py (Part 1)

```python
# import datetime
# datetime.time
from datetime import time

start = time()
print(start)
print(type(start))
print(start.hour)
print(start.minute)
print(start.second)

print(time(6))
print(time(hour = 6))
print(time(hour = 18))
```

the-time-object.py (Part 2)

```python
print(time(12, 25))
print(time(hour = 12, minute = 25))

# 11:34:22PM
print(time(23, 34, 22))
evening = time(hour = 23, minute = 34, second = 22)
print(evening.hour)
print(evening.minute)
print(evening.second)

# time(27)
```

the-datetime-object-I.py (Part 1)

```
from datetime import datetime

# import datetime
# datetime.datetime

print(datetime(1999, 7, 24))
print(datetime(1999, 7, 24, 14))
print(datetime(1999, 7, 24, 14, 16))
print(datetime(1999, 7, 24, 14, 16, 58))
print(datetime(year = 1999, month = 7, day = 24, hour = 14, minute = 16, second =
58))

today = datetime.today()
```

the-datetime-object-I.py (Part 2)

```
print(today)
print(datetime.now())
print(today.year)
print(today.month)
print(today.day)
print(today.hour)
print(today.minute)
print(today.second)

print(today.weekday())

same_time_twenty_years_ago = today.replace(year = 1999)
print(same_time_twenty_years_ago)

same_time_in_january = today.replace(month = 1)
print(same_time_in_january)

start_of_january_1999 = today.replace(year = 1999, month = 1, day = 1)
print(start_of_january_1999)
```

the-datetime-object-II.py

```
from datetime import datetime

today = datetime.today()

print(today.strftime("%m"))
print(today.strftime("%m %d"))
print(today.strftime("%m/%d/%Y"))
print(today.strftime("%m-%d-%Y"))

print(today.strftime("%Y-%m-%d"))
print(today.strftime("%y-%m-%d"))

print(today.strftime("%A"))
print(today.strftime("%B"))
```

the-timedelta-object.py

```
from datetime import datetime, timedelta

birthday = datetime(1991, 4, 12)
today = datetime.now()

my_life_span = today - birthday
print(my_life_span)
print(type(my_life_span))

print(my_life_span.total_seconds())

five_hundred_days = timedelta(days = 500, hours = 12)
print(five_hundred_days)

print(five_hundred_days + five_hundred_days)

print(today + five_hundred_days)
```

The random Module

the-random-randint-and-randrange-functions.py

```python
import random

print(random.random() * 100)

print(random.randint(1, 5))

print(random.randrange(0, 50, 10))
```

the-choice-and-sample-functions.py

```python
import random

print(random.choice(["Bob", "Moe", "Curly"]))
print(random.choice((1, 2, 3)))
print(random.choice("elephant"))

lottery_numbers = [random.randint(1, 50) for value in range(50)]
# print(lottery_numbers)

print(random.sample(lottery_numbers, 1))
print(random.sample(lottery_numbers, 2))
print(random.sample(lottery_numbers, 6))
```

the-shuffle-function.py

```
import random
import copy

characters = ["Warrior", "Druid", "Hunter", "Rogue", "Mage"]

clone = characters[:]
clone = characters.copy()
clone = copy.copy(characters)

random.shuffle(clone)

print(characters)
print(clone)
```

30

Testing Code: The Basics

the-assert-statement.py

```
def add(x, y):
 assert isinstance(x, int) and isinstance(y, int), "Both arguments must be
integers"
 return x + y

print(add(3, 5))
# print(add(3, "5"))
```

the-doctest-module.py

```python
def sum_of_list(numbers):
    """Return the sum of all numbers in a list.
    >>> sum_of_list([1, 2, 3])
    6
    >>> sum_of_list([5, 8, 13])
    26
    """
    total = 0
    for num in numbers:
        total += num
    return total

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

the-unittest-module.py

```python
import unittest

class TestStringMethods(unittest.TestCase):
    def test_split(self):
        pass

if __name__ == "__main__":
    unittest.main()
```

the-assertEqual-method.py

```python
import unittest

class TestStringMethods(unittest.TestCase):
   def test_split(self):
       self.assertEqual("a-b-c".split("-"), ["a", "b", "c"])
       self.assertEqual("d+e+f".split("+"), ["d", "e", "f"])

   def test_count(self):
       self.assertEqual("beautiful".count("u"), 2)

if __name__ == "__main__":
   unittest.main()
```

the-purpose-of-testing.py

```python
import unittest

def multiply(a, b):
    return a * b

class MultiplyTestCase(unittest.TestCase):
    def test_multiply(self):
        self.assertEqual(multiply(3, 4), 12)

if __name__ == "__main__":
    unittest.main()
```

skipping-tests.py

```python
import unittest

class TestSkippingStuff(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 1, 2)

    def test_subtraction(self):
        self.assertEqual(10 - 5, 5)

    @unittest.skip("To be implemented later")
    def test_multiplication(self):
        pass

if __name__ == "__main__":
    unittest.main()
```

assert-not-equal-and-custom-error-messages.py (part 1)

```python
import unittest

def copy_and_add_element(values, element):
    copy = values
    copy.append(element)
    return copy
```

assert-not-equal-and-custom-error-messages.py (Part 2)

```
class TestInequality(unittest.TestCase):
   def test_inequality(self):
       self.assertNotEqual(1, 2)
       self.assertNotEqual(True, False)
       self.assertNotEqual("Hello", "hello")
       self.assertNotEqual([1, 2], [2, 1])

   def test_copy_and_add_element(self):
       values = [1, 2, 3]
       result = copy_and_add_element(values, 4)

       self.assertEqual(result, [1, 2, 3, 4])

       self.assertNotEqual(
           values,
           [1, 2, 3, 4],
           "The copy_and_add_element function is mutating the input. Make sure you're creating a copy."
       )

if __name__ == "__main__":
   unittest.main()
```

object-identity.py

```
import unittest

class IdentityTests(unittest.TestCase):
   def test_identicality(self):
       a = [1, 2, 3]
       b = a
       c = [1, 2, 3]

       self.assertEqual(a, b)
       self.assertEqual(a, c)

       self.assertIs(a, b)
       self.assertIsNot(a, c)
       self.assertIsNot(b, c)

if __name__ == "__main__":
   unittest.main()
```

truthiness-and-falsiness.py

```python
import unittest

class TruthinessAndFalsinessTests(unittest.TestCase):
   def test_truthiness(self):
       # self.assertEqual(3 < 5, True)
       self.assertTrue(3 < 5)
       self.assertTrue(1)
       self.assertTrue("hello")
       self.assertTrue(["a"])
       self.assertTrue({ "b": 5 })

   def test_falsiness(self):
       self.assertFalse(False)
       self.assertFalse(0)
       self.assertFalse("")
       self.assertFalse([])
       self.assertFalse({})

if __name__ == "__main__":
   unittest.main()
```

nullness.py

```python
import unittest

def explicit_return_sum(a, b):
   return a + b

def implicit_return_sum(a, b):
   print(a + b)

class TestNone(unittest.TestCase):
   def test_sum_functions(self):
       self.assertIsNone(implicit_return_sum(3, 5))
       self.assertIsNotNone(explicit_return_sum(10, 2))

if __name__ == "__main__":
   unittest.main()
```

inclusion.py

```python
import unittest
class InclusionTests(unittest.TestCase):
   def test_inclusion(self):
       # self.assertTrue("k" in "king")
       self.assertIn("k", "king")
       self.assertIn(1, [1, 2, 3])
       self.assertIn(5, (6, 5, 7))
       self.assertIn("a", { "a": 1, "b": 2})
       self.assertIn("a", { "a": 1, "b": 2}.keys())
       self.assertIn(2, { "a": 1, "b": 2}.values())
       self.assertIn(55, range(50, 59))

   def test_non_inclusion(self):
       self.assertNotIn("w", "king")
       self.assertNotIn(10, [1, 2, 3])
       self.assertNotIn(15, (6, 5, 7))
       self.assertNotIn("c", { "a": 1, "b": 2})
       self.assertNotIn("c", { "a": 1, "b": 2}.keys())
       self.assertNotIn(5, { "a": 1, "b": 2}.values())
       self.assertNotIn(65, range(50, 59))

if __name__ == "__main__":
   unittest.main()
```

object-type.py

```python
import unittest

class ObjectTypeTests(unittest.TestCase):
   def test_is_instance(self):
       self.assertIsInstance(1, int)
       self.assertIsInstance(8.765, float)
       self.assertIsInstance([], list)
       self.assertIsInstance({ "a": 1 }, dict)
       # self.assertIsInstance({ "a": 1 }, list)

   def test_not_is_instance(self):
       self.assertNotIsInstance(5, list)
       self.assertNotIsInstance(5, float)
       self.assertNotIsInstance(5, set)
       self.assertNotIsInstance(5, dict)
       # self.assertNotIsInstance(5, int)

if __name__ == "__main__":
   unittest.main()
```

testing-errors.py

```
import unittest

def divide(x, y):
   if y == 0:
       raise ZeroDivisionError
   return x / y

class DivideTestCase(unittest.TestCase):
   def test_divide(self):
       self.assertRaises(ZeroDivisionError, divide, 10, 0)

   def test_divide_another_way(self):
       with self.assertRaises(ZeroDivisionError):
           divide(10, 0)

if __name__ == "__main__":
   unittest.main()
```

setup-and-teardown.py (Part 1)

```python
import unittest
class Address():
   def __init__(self, city, state):
       self.city = city
       self.state = state

class Owner():
   def __init__(self, name, age):
       self.name = name
       self.age = age

class Restaurant():
   def __init__(self, address, owner):
       self.address = address
       self.owner = owner

   @property
   def owner_age(self):
       return self.owner.age

   def summary(self):
       return f"This restaurant is owned by {self.owner.name} and is located in {self.address.city}."
```

setup-and-teardown.py (Part 2)

```python
class TestRestaurant(unittest.TestCase):
   def setUp(self):
       print("This will run before each test!")
       address = Address(city = "New York", state = "New York")
       owner = Owner(name = "Jackie", age = 60)
       self.golden_palace = Restaurant(address, owner)

   def tearDown(self):
       print("This will run after each test!")

   def test_owner_age(self):
       self.assertEqual(self.golden_palace.owner_age, 60)

   def test_summary(self):
       self.assertEqual(
           self.golden_palace.summary(),
           "This restaurant is owned by Jackie and is located in New York."
       )

if __name__ == "__main__":
   unittest.main()
```

setUpClass-and-tearDownClass.py

```
import unittest
class TestOperations(unittest.TestCase):
   @classmethod
   def setUpClass(cls):
       print("This will run ONCE before the test suite starts")

   def setUp(self):
       print("This will run before EACH test")

   def tearDown(self):
       print("This will run after EACH test")

   @classmethod
   def tearDownClass(cls):
       print("This will run ONCE after the test suite finishes")

   def test_stuff(self):
       self.assertEqual(1, 1)

   def test_more_stuff(self):
       self.assertEqual([], [])

if __name__ == "__main__":
   unittest.main()
```

Testing Code: Mocking

intro-to-mocking-with-the-Mock-class.py

```python
from unittest.mock import Mock

pizza = Mock()
print(pizza)

print(type(pizza))

pizza.configure_mock(size = "Large", price = 19.99, toppings = ["Pepperoni", "Mushroom", "Sausage"])
# pizza.size = "Large"
# pizza.price = 19.99
# pizza.toppings = ["Pepperoni", "Mushroom", "Sausage"]

print(pizza.size)
print(pizza.price)
print(pizza.toppings)

# print(pizza.anything)
# print(pizza.anything.we.want)
print(pizza.cover_with_cheese())
print(pizza.cover_with_cheese())
```

the-return-value-attribute.py

```python
from unittest.mock import Mock

mock = Mock(return_value = 25)

# print(mock.return_value)
# mock.return_value = 25
print(mock())

stuntman = Mock()
stuntman.jump_off_building.return_value = "Oh no, my leg"
stuntman.light_on_fire.return_value = "It burns!"

print(stuntman.jump_off_building())
print(stuntman.light_on_fire())
```

the-side-effect-attribute.py

```python
from unittest.mock import Mock
from random import randint

def generate_number():
   return randint(1, 10)

call_me_maybe = Mock(return_value = 10, side_effect = generate_number)
# call_me_maybe.side_effect = generate_number
print(call_me_maybe())
print(call_me_maybe())
print(call_me_maybe())

three_item_list = Mock()
three_item_list.pop.side_effect = [3, 2, 1, IndexError("pop from empty list")]
print(three_item_list.pop())
print(three_item_list.pop())
print(three_item_list.pop())
# print(three_item_list.pop())

mock = Mock(side_effect = NameError("Some error message"))
mock.side_effect = None
print(mock())
```

Mock-vs-MagicMock-objects.py (Part 1)

```python
from unittest.mock import Mock, MagicMock

plain_mock = Mock()
magic_mock = MagicMock()

# print(len(plain_mock)) # __len__
print(len(magic_mock))

# print(plain_mock[3])

print(magic_mock[3])
print(magic_mock[100])
print(magic_mock["hello"])
# __getitem__
```

Mock-vs-MagicMock-objects.py (Part 2)

```
magic_mock.__len__.return_value = 50
print(len(magic_mock))

if magic_mock:
   print("hello")

magic_mock.__bool__.return_value = False

if magic_mock:
   print("goodbye")

magic_mock.__getitem__.return_value = 100
print(magic_mock[3])
print(magic_mock[100])
print(magic_mock["hello"])
```

mock-calls.py (Part 1)

```python
import unittest
from unittest.mock import MagicMock

class MockCallsTest(unittest.TestCase):
    def test_mock_calls(self):
        mock = MagicMock()
        mock()
        # mock()
        # mock()
        mock.assert_called()

    def test_not_called(self):
        mock = MagicMock()
        # mock()
        mock.assert_not_called()
```

mock-calls.py (Part 2)

```
    def test_called_with(self):
        mock = MagicMock()
        mock(1, 2, 3)
        mock.assert_called_with(1, 2, 3)

    def test_mock_attributes(self):
        mock = MagicMock()
        mock()
        mock(1, 2)
        print(mock.called)
        print(mock.call_count)
        print(mock.mock_calls)

if __name__ == "__main__":
    unittest.main()
```

putting-it-all-together.py (Part 1)

```python
import unittest
from unittest.mock import MagicMock

class Actor():
   def jump_out_of_helicopter(self):
       return "Nope, not doing it!"

   def light_on_fire(self):
       return "Heck no, where's my agent?"

class Movie():
   def __init__(self, actor):
       self.actor = actor

   def start_filming(self):
       self.actor.jump_out_of_helicopter()
       self.actor.light_on_fire()
```

putting-it-all-together.py (Part 1)

```
class MovieTest(unittest.TestCase):
   def test_start_filming(self):
       stuntman = MagicMock()
       movie = Movie(stuntman)

       movie.start_filming()
       stuntman.jump_out_of_helicopter.assert_called()
       stuntman.light_on_fire.assert_called()

if __name__ == "__main__":
   unittest.main()
```

verifying-doubles.py (Part 1)

```python
from unittest.mock import MagicMock

class BurritoBowl():
    restaurant_name = "Bobo's Burritos"

    @classmethod
    def steak_special(cls):
        return cls("Steak", "White", 1)

    def __init__(self, protein, rice, guacamole_portions):
        self.protein = protein
        self.rice = rice
        self.guacamole_portions = guacamole_portions

    def add_guac(self):
        self.guacamole_portions += 1
```

verifying-doubles.py (Part 2)

```
# lunch = BurritoBowl.steak_special()
# print(lunch.protein)
# lunch.add_guac()
# print(lunch.guacamole_portions)
class_mock = MagicMock(spec = BurritoBowl)
print(class_mock.restaurant_name)
print(class_mock.steak_special())
# print(class_mock.chicken_special())
# print(class_mock.city)
instance_mock = MagicMock(spec_set = BurritoBowl.steak_special())
print(instance_mock.protein)
print(instance_mock.rice)
print(instance_mock.guacamole_portions)
print(instance_mock.add_guac())
# print(instance_mock.add_cheese())
# print(instance_mock.beans)
# instance_mock.beans = True
# print(instance_mock.beans)
```

patch-l.py (Part 1)

```python
import urllib.request
import unittest
from unittest.mock import patch

class WebRequest():
    def __init__(self, url):
        self.url = url

    def execute(self):
        response = urllib.request.urlopen(self.url)
        if response.status == 200:
            return "SUCCESS"

        return "FAILURE"

# wr = WebRequest("http://www.google.com")
# wr.execute()
```

patch-l.py (Part 2)

```python
class WebRequestTest(unittest.TestCase):
   def test_execute_with_success_response(self):
       with patch('urllib.request.urlopen') as mock_urlopen:
           mock_urlopen.return_value.status = 200
           wr = WebRequest("http://www.google.com")
           self.assertEqual(wr.execute(), "SUCCESS")

   def test_execute_with_failure_response(self):
       with patch('urllib.request.urlopen') as mock_urlopen:
           mock_urlopen.return_value.status = 404
           wr = WebRequest("http://www.google.com")
           self.assertEqual(wr.execute(), "FAILURE")


if __name__ == "__main__":
   unittest.main()
```

patch-II.py (Part 1)

```python
import urllib.request
import unittest
from unittest.mock import patch

class WebRequest():
   def __init__(self, url):
       self.url = url

   def execute(self):
       response = urllib.request.urlopen(self.url)
       if response.status == 200:
           return "SUCCESS"

       return "FAILURE"
```

patch-II.py (Part 2)

```python
class WebRequestTest(unittest.TestCase):
    @patch('urllib.request.urlopen')
    def test_execute_with_success_response(self, mock_urlopen):
        mock_urlopen.return_value.status = 200
        wr = WebRequest("http://www.google.com")
        self.assertEqual(wr.execute(), "SUCCESS")

    @patch('urllib.request.urlopen')
    def test_execute_with_failure_response(self, mock_urlopen):
        mock_urlopen.return_value.status = 404
        wr = WebRequest("http://www.google.com")
        self.assertEqual(wr.execute(), "FAILURE")

if __name__ == "__main__":
    unittest.main()
```

patch-III.py

```python
import unittest
from unittest.mock import patch
from web_request import WebRequest

class WebRequestTest(unittest.TestCase):
   @patch('web_request.urlopen')
   def test_execute_with_success_response(self, mock_urlopen):
       mock_urlopen.return_value.status = 200
       wr = WebRequest("http://www.google.com")
       self.assertEqual(wr.execute(), "SUCCESS")

   @patch('web_request.urlopen')
   def test_execute_with_failure_response(self, mock_urlopen):
       mock_urlopen.return_value.status = 404
       wr = WebRequest("http://www.google.com")
       self.assertEqual(wr.execute(), "FAILURE")

if __name__ == "__main__":
   unittest.main()
```