Code Examples and Teaching Instructions Python Basics

Contents

[Variables 2](#_Toc66043652)

[Strings 5](#_Toc66043653)

[Lists 9](#_Toc66043654)

[Tuples 11](#_Toc66043655)

[Dictionaries 12](#_Toc66043656)

[Sets 14](#_Toc66043657)

[Conditionals 16](#_Toc66043658)

[Input and Try/Catch 19](#_Toc66043659)

[Functions 20](#_Toc66043660)

[Classes 23](#_Toc66043661)

[Loops 26](#_Toc66043662)

[Datetime 29](#_Toc66043663)

[File IO 30](#_Toc66043664)

[Random 31](#_Toc66043665)

[Command Line Arguments 32](#_Toc66043666)

[Json 33](#_Toc66043667)

[CSV 35](#_Toc66043668)

[Modules 37](#_Toc66043669)

# Variables

# Introduction to Python Basics

# Python Variables

# --------------------------

# No variable type declaration, variable is created when you assign it

# Basic types: Int, Float, String (single or double quotes), Boolean

# Strings can be represented using single or double quotes

1

1.0

"double quotes"

'single quotes'

True

False

# Variables are case sensitive e.g. Name <> name

# Begin with letter or \_ and may contain letters, numbers and underscore

# Conventions says to use Snake Case naming e.g. my\_variable\_name = "John"

a = 3

b = 4.0

c = 'abcde'

d = True

# You can assign multiple variables to different values at once

x, y, z = 1, 'Two', 3.0

# You can assign multiple variables to the same value at once

u = v = w = 'Same'

# Variables can be changed after first assigned

f = 1

f = 'change f to a string'

# Printing to the console

# --------------------------

# Use print()

# You can print variables, literals, objects

print(a)

print(b)

print(c)

print(d)

print('Hello World')

# You can print multiple things at once using a comma splice

print(x, y, z)

# You can specify what is printed af the end of the print statement

print(a, end=' [[Print this at the end]]')

print()

print(a, end='\n')

# Comments

# --------------------------

# Use the hashtag to create a single comment e.g. #

# Use triple quotes to create a block comment e.g. """ """

# This is an example of a one line comment

"""

    This is an example of a block commment

    that can span across multiple new lines

"""

# Casting variables

# --------------------------

# You can specify a variables type by casting e.g. int(), str(), float(), bool()

m = int(1)

n = str(1)

o = float(1)

p = bool(1)

q = bool('True')

print(m, n, o, p, q)

# Getting the variable type

# --------------------------

# Use type() to retrieve the type

r = 1

s = 1.0

t = '1'

u = True

print(type(r))

print(type(s))

print(type(t))

print(type(u))

# Strings

# Introduction to String Basics

# A string in Python is a sequence of characters.

# String literals can be created using single, double or triple quotes

# --------------------------

a = 'This uses single quotes'

b = "This uses double quotes"

c = """

    This uses triple quotes

    so it can span multiple lines.

"""

print(a)

print(b)

print(c)

# Format a string

# --------------------------

# String Modulo Operator

# e.g. <format\_string> % <values>

# On the left side of the % operator, <format\_string> is a string containing

# one or more conversion specifiers.

# The <values> on the right side get inserted into <format\_string>

# in place of the conversion specifiers.

# The resulting formatted string is the value of the expression

print('%d %s cost $%.2f' % (6, 'bananas', 1.74))

# String .format method (similar to modulo operator)

print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))

print('{quantity} {item} cost ${price}'.format(quantity=6, item='bananas', price=1.74))

# the Formatted String Literal using f-string

quantity = 6

item = 'bananas'

price = 1.74

print(f'{quantity} {item} cost ${price}')

# Concatenation and string conversion

print(str(quantity) + ' ' + item + ' cost $' + str(price))

# string literals next to each other are concatenated

print('These' 'words' 'are' 'automatically' 'concantenated')

# Use single quotes inside of double quotes

# or double quotes inside of single quotes

print("I can't stop using contractions")

print('I once said, "Quotes are great"')

# Get the length of a string

print(len(a))

print(len(b))

print(len(c))

# Remove white space in a string

d = "  Remove the spaces   "

d1 = d.strip()

d2 = d.lstrip()

d3 = d.rstrip()

print(f"[{d1}]")

print(f"[{d2}]")

print(f"[{d3}]")

# Repeating strings

# --------------------------

r = 'repeat '

print(r \* 5)

print('-' \* 50)

# Accessing string elements

# --------------------------

# A string is simply a list of characters that can be retrieved using an index

# the index of a string starts at zero

s = 'abcdefg'

print(s[0])

print(s[4])

# When the index is negative, we retrieve elements from the end of the string

print(s[-1])

print(s[-2])

# We can retrieve a range of characters

# Indicate the start:end:step when retrieving a range

print(s[0:4])   # first 4 chars

print(s[2:6])   # chars from position 2 through position 6

print(s[:])     # full string

print(s[::-1])  # reverse the string

# Finding substrings

# --------------------------

# find() and rfind() return the index of the first occurrence of the substring in the string

# or they return -1 if not found

fruits = 'apple banana watermelon banana'

print(fruits.find('banana'))            # search from the start of the string

print(fruits.rfind('banana'))           # search from the end of the string

print(fruits.find('banana', 10))        # start searching at position 10

print(fruits.find('banana', 10, 20))    # start searching at position 10 through position 20

# Replacing strings

# --------------------------

# replace() replaced substrings in a string with other substrings

old = 'My name is Fred'

new = old.replace('Fred', 'Johnny')

print(new)

t = 'I saw a fox. The fox was a good fox.'

t1 = t.replace('fox', 'dog')

print(t1)

t2 = t.replace('fox', 'dog', 2) # specifiy how many occurances to replace

print(t2)

# Splitting and joining strings

# --------------------------

# split() will return a list of string cut using a separator. By default it splits on spaces.

abc = 'a b c d e f g h i'

letters = abc.split()

print(letters)

nums = '1,2,3,4,5,6,7,8,9'

values = nums.split(',')

print(values)

# join() will create one string from a list of strings separated by a given character

j = ':'.join(letters)

print(j)

# String case

# --------------------------

# Change the case of a string

v = 'UPPER lower'

print(v.upper())

print(v.lower())

print(v.swapcase())

print(v.title())

# Lists

# Lists

# Used to store multiple items as a collection

# of data in a single variable

# List items are ordered, changeable, and allow duplicate values

# list indexes are zero based meaning they start at 0 not 1

# --------------------------

mylist = ["apple", "banana", "cherry", "peach", "banana"]

print(mylist)

various = [1, 2, True, 'Bob', 1.25]

print(various)

print(mylist[2])        # element of a list

print(mylist[:1])       # range  of a list

print(mylist[1:3])

print(mylist[::-1])     # reverse a list

various.append('New')   # add to a list

print(various)

various.pop()           # remove last item from list

print(various)

various.pop(1)           # remove specific index from list

print(various)

mylist.remove('cherry')    # remove specific item from list

print(mylist)

del mylist[0]           # remove specific index from list

print(mylist)

print(len(mylist))      # length of a list

mylist[0] = "berry"

print(mylist)

# Using the list constructor

# --------------------------

thislist = list()

thislist.append('Red')

thislist.append('White')

thislist.append('Blue')

print(thislist)

# Searching a list

# --------------------------

print(mylist.index("banana"))   # Find location of value in a list

print(mylist.count("banana"))   # Count occurences of value in a list

print("banana" in mylist)       # Is a value in the list

# Copying a list

# --------------------------

# lists default copy by reference

newlist = mylist

print(mylist)

print(newlist)

mylist.append("grape")

print(mylist)

print(newlist)

# use copy command to get an actual copy

copylist = mylist.copy()

mylist.append("pineapple")

print(mylist)

print(copylist)

# Tuples

# Tuples

# Tuples are used to store multiple items in a single variable.

# A tuple is a collection which is ordered, unchangeable and allow duplicate values

# --------------------------

tuple1 = ("apple", "banana", "cherry")

print(tuple1)

# You can create a tuple without parenthesis as well

tuple2 = 1, 2, "Bob", True

print(tuple2)

# Using the tuple constructor

# --------------------------

tuple3 = tuple(('red', 'blue'))

print(tuple3)

# Tuple with only one item

# --------------------------

tuple4 = ('No') # this is not a tuple

print(tuple4)

tuple5 = ('Yes',) # this is a tuple when it has a trailing comma

print(tuple5)

print(len(tuple1))      # length of a tuple

# Use tuples to unpack multiple values

# --------------------------

v1, v2, v3 = ('Scott', 25, True)

print(v1, v2, v3)

# Dictionaries

# Dictionaries

# Dictionaries are used to store data values in key:value pairs.

# A dictionary is a collection which is ordered\*, changeable and does not allow duplicates.

# Dictionary iteams are presetned in key:value pairs and ben be referred to using the key name

# --------------------------

dict1 = {

    'brand': 'Ford',

    'model': 'Mustang',

    'year': 1964

}

print(dict1)

print(dict1['brand'])   # Print the value of a specific key

print(dict1['model'])

# change the value of a key

dict1['model'] = 'Thunderbird'

print(dict1['model'])

#dDuplicate keys are not permitted, the last value will overrite the existing values

dict2 = {

    'street': '13 Main Street',

    'city': 'Kearns',

    'city': 'Sandy'

}

print(dict2)

print(len(dict2))      # length of a dictionary

# Using the dict constructor

# --------------------------

dict3 = dict()

dict3['name'] = 'Bob'

dict3['age'] = 25

print(dict3)

# Dict can hold list, tuples, sets, and dictionaries

# --------------------------

dict4 = {

    'colors': ('red', 'white', 'blue'),

    'point': (10,21),

    'car': {'make': 'Dodge', 'model': 'Charger'}

}

print(dict4)

# Numbers as keys

# --------------------------

dict5 = {

    1: 'zombie',

    2: 'vampire',

    3: 'werewolf',

    4: 'ghost'

}

print(dict5)

# Sets

# Sets

# Sets are used to store multiple items in a single variable.

# A set is a collection which is both unordered and unindexed.

# --------------------------

set1 = {"apple", "banana", "cherry"}

print(set1)

# A set does not allow duplicate values

# --------------------------

set2 = {"apple", "banana", "cherry", "banana"}

print(set2)

print(len(set1))      # length of a set

print(len(set2))

# A set can contain multiple types and other hashable types such as a tuple

# --------------------------

set3 = {

    "Bob",

    25,

    ("red", "white", "blue"),

}

print(set3)

# Using the set constructor, note we use the parenthesis and not the curly braces

# --------------------------

set4 = set((1,2,3,4,3,5,7,2,1))

print(set4)

# Using the set() constructor to make a set of unique characters

# --------------------------

set5 = set('Hello, my name is Bob')

print(set5)

# Union and Intersection of sets

# --------------------------

setA = set('1234567')

setB = set('12389')

setC = set('123')

print(setA.intersection(setB))

print(setA.union(setB))

print(setA.issuperset(setB))

print(setA.issuperset(setC))

print(setA.difference(setB))

print(setA-setB)

# Conditionals

# Conditions and if statements

# Python supports the usual logical conditions from mathematics:

'''

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

'''

# The if statement

# --------------------------

a = 1

b = 2

if a < b:

    print(f"{a} is less than {b}")

# Indentation

# Python relies on indentation (whitespace at the beginning of a line)

# to define scope in the code.

# Other programming languages often use curly-brackets for this purpose.

# the same if statement without indentation will raise an error

'''

if a < b:

print(f"{a} is less than {b}")

'''

# would raise an error

# The elif statement

# The elif keyword is pythons way of saying

# "if the previous conditions were not true, then try this condition".

# --------------------------

a = 33

b = 20

if a < b:

    print(f"{a} is less than {b}")

elif a > b:

    print(f"{a} is greater than {b}")

# The else statement

# The else keyword catches anything which isn't caught by the preceding conditions.

# --------------------------

a = 40

b = 40

if a < b:

    print(f"{a} is less than {b}")

elif a > b:

    print(f"{a} is greater than {b}")

else:

    print(f"{a} must be equal to {b}")

# The ternary statement

# The else keyword catches anything which isn't caught by the preceding conditions.

# --------------------------

a = 40

b = 41

print(f"{a} is less than {b}") if a < b else print(f"{a} is not less than {b}")

# The ternary assignment statement

min = a if a < b else b

print(f"the smaller value is {min}")

# Logical operators

# --------------------------

# The and keyword is a logical operator, and is used to combine conditional statements

a, b, c = 200, 33, 500

if a > b and c > a:

    print("Both conditions are true")

# The or keyword is a logical operator, and is used to combine conditional statements

a, b, c = 200, 33, 500

if a > b or a > c:

  print("At least one of the conditions is True")

# Nested if statements

# --------------------------

x = 41

if x > 10:

  print("Above ten,")

  if x > 20:

    print("and also above 20!")

  else:

    print("but not above 20.")

# The pass statement

# if statements cannot be empty, but if you for some reason have an if statement with no content,

# put in the pass statement to avoid getting an error.

# --------------------------

a = 33

b = 200

if b > a:

  pass

# Input and Try/Catch

# Input and try/except

num1 = input()

print(num1)

try:

    print("Welcome to the add program")

    n1 = int(input("Enter the first number: "))

    n2 = input("Enter the second number: ")

    print(f"The added numbers are: {n1 + int(n2)}")

except:

    print("Please only enter numbers")

# Functions

# Functions

# A function is a block of code which only runs when it is called.

# You can pass data, known as parameters, into a function.

# A function can return data as a result.

# Creating a function

# --------------------------

# functions use the def keyword then the function name followed by a colon.

# note how the body of the functino will be indented

def my\_function():

  print("Hello from a function")

# Calling a function

# --------------------------

my\_function()

# Return values

# --------------------------

def my\_function\_returns():

    return 'Thank you for calling this function'

print(my\_function\_returns())

# Arguments

# --------------------------

# Information can be passed into functions as arguments.

# Arguments are specified after the function name, inside the parentheses.

# You can add as many arguments as you want, just separate them with a comma.

def my\_function2(name): # single parameter

  print(f"Hi, my name is {name}")

my\_function2("Bob")

my\_function2("Suzi")

def my\_function3(first, last):  # multiple parameters

    print(f'My name is {first} {last}')

my\_function3('Bob', 'Thomas')

# Arbitrary arguments

# If the number of arguments is unknown, add a \* before the parameter name

# --------------------------

def my\_function4(\*kids):

    print(f"I have {len(kids)} kids")

    print(f'Kid #2 is named {kids[1]}')

my\_function4('Tom', 'Sally', 'Bruce')

# Keyword Arguments

# You can also send arguments with the key = value syntax.

# --------------------------

def my\_function5(child3, child2, child1):

  print("The youngest child is " + child3)

my\_function5(child1 = "Tom", child2 = "Sally", child3 = "Bruce")

# Arbitrary Keyword Arguments

# If you do not know how many keyword arguments that will be passed into your function,

# add two asterisk: \*\* before the parameter name in the function definition.

# --------------------------

def my\_function6(\*\*kid):

  print(f"The youngest child is {kid['child3']}")

my\_function5(child1 = "Tom", child2 = "Sally", child3 = "Bruce")

# Default Parameter Value

# If we call the function without argument, it uses the default value

# --------------------------

def my\_function7(age = 25):

  print(f"I am {age} years old")

my\_function7(20)

my\_function7()

# Passing a other types as an argument

# You can send any data types of argument to a function (string, number, list, dictionary etc.),

# and it will be treated as the same data type inside the function.

# --------------------------

def my\_function8(fruit):

  print(f"I love {len(fruit)} kinds of fruit.  They are {fruit}")

fruits = ["apple", "banana", "cherry"]

my\_function8(fruits)

# Global variables

# You can designate a variable initialized outside of a functino

# to be used globally inside the function with the global key word

outside = 100

def my\_function9():

    global outside

    outside += 10

my\_function9()

print(outside)

# Recursion

# Recursion is a common programming concept. It means that a function calls itself.

# This has the benefit that you can loop through data to reach a result.

# --------------------------

loops = 0

def try\_recursion(value):

    global loops

    loops += 1

    print(f'looping: {loops}')

    if value > 0:

        result = value + try\_recursion(value - 1)

        print(result)

    else:

        result = 0

    return result

try\_recursion(6)

# Classes

# Classes and Objects

from os import system

system('cls')           # neat trick to clear the screen when you execute the code

# Python is an object oriented programming language.

# Almost everything in Python is an object, with its properties and methods.

# A Class is like an object constructor, or a "blueprint" for creating objects.

# Creating a Class and an object

# --------------------------

class MyClass:

    prop = True

m1 = MyClass()  # an instance of the class creates and object

print(m1.prop)

# The \_\_init\_\_() function

# --------------------------

# All classes have a function called \_\_init\_\_(), which is the constructor of the class.

# This is always executed when the class is being initiated.

# Use the \_\_init\_\_() function to assign values to object properties,

# or other operations that are necessary to do when the object is being created

class Person:

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

p1 = Person("Fred", 25)

print(p1.name, p1.age)

# The self parameter is a reference to the current instance of the class,

# and is used to access variables that belongs to the class.

# It does not have to be named self, you can call it whatever you like,

# but it has to be the first parameter of any function in the class:

# Class methods

# --------------------------

# Classes can also contain methods. Methods in objects are functions that will belong to the object.

class Animal:

    def \_\_init\_\_(self, name, type):

        self.name = name

        self.type = type

        self.is\_alive = True

    def introduce(self):

        print(f"Hello my {self.type}'s name is {self.name}")

a1 = Animal('Fido', 'dog')

a1.introduce()

a1.name = 'Rover'

a1.introduce()  # You can modify object properties

print(a1.is\_alive)

del a1.is\_alive # you can delete an object property

del a1          # you can delete an object

# Classes Inheritence and Polymorphism

# --------------------------

# Inheritance - allows us to define a class that inherits all the methods and properties from another class.

# Polymorphism - allows us to change the form of a property or method to meet the needs of a subclass

class bird():

    def \_\_init\_\_(self):

        print('bird is initialized')

        self.color = 'grey'

    def flight(self):

        print('Most of the birds can fly but some cannot')

class sparrow(bird):    # Inheriting bird

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.color = 'red brown'

        print('sparrow is inialized')

    def flight(self):   # Polymorphing the flight method for sparrows

        print('Sparrows can fly')

class ostrich(bird):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.color = 'white brown'

        print('ostrich is inialized')

    def flight(self):

        print('Ostriches cannot fly')

my\_sparrow = sparrow()

print(my\_sparrow.color)

my\_sparrow.flight()

my\_ostrich = ostrich()

print(my\_ostrich.color)

my\_ostrich.flight()

# Loops

# Loops

# Python has two primitive loop commands:

#   while loops

#   for loops

# The while loop

# --------------------------

# With the while loop we can execute a set of statements as long as a condition is true.

i = 1

while i < 6:

  print(i)

  i += 1

# Breaking a loop

# With the break statement we can stop the loop even if the while condition is true

i = 1

while i < 6:

  print(i)

  if i == 3:

    break

  i += 1

# Continuing in a loop

# With the continue statement we can stop the current iteration, and continue with the next

i = 0

while i < 6:

  i += 1

  if i == 3:

    continue

  print(i)

# Using the else statement

i = 1

while i < 6:

  print(i)

  i += 1

else:

  print("i is no longer less than 6")

# The for loop

# --------------------------

# A for loop is used for iterating over a sequence

# (that is either a list, a tuple, a dictionary, a set, or a string).

# With the for loop we can execute a set of statements,

# once for each item in a list, tuple, set etc.

fruits = ["apple", "banana", "cherry"]

for f in fruits:

  print(f)

for i in 'bananas':

    print(i)

# Iterating over a range

# To loop through a set of code a specified number of times, we can use the range() function,

for i in range(6):

  print(i)

for i in range(2, 6): # begin, end

  print(i)

for i in range(2, 6, 3): # begin, end, step

  print(i)

# Reverse direction

# --------------------------

for i in reversed(range(6)):

  print(i)

# Nested loops

# --------------------------

adj = ["red", "big", "tasty"]

fruits = ["apple", "banana", "cherry"]

for a in adj:

  for f in fruits:

    print(a, f)

# Datetime

# Date and time

import datetime

todaysDate = datetime.date.today()

print(todaysDate)

print(todaysDate.month)

niceDate = f"{todaysDate.month}/{todaysDate.day}/{todaysDate.year}"

print(niceDate)

print(type(todaysDate))

# File IO

# File I/O

# 'r'   read            Stream positioned at the beginning of file

# 'w'   write           File created if it doens't exist. Stream positioned at beginning of file

# 'a'   append          File created if it doesn't exist. Stream positioned at end of file

# 'r+'  read and write  Stream positioned at the beginning of the file

# 'w+'  read and write  File created if it doesn't exist. Stream positioned at begining of file

# 'a+'  read adn write  File created if it doesn't exist. Stream positioned at end of file

file = open('test1.txt', 'w')

file.write("Hello\n")

file.write("this is a test\n")

file.close()

file = open('test1.txt', 'r')

content = file.read()

file.close()

print(content)

file = open('test1.txt', 'a')

file.write('add one more line to this file\n')

file.close()

# open files using "with"

#-----------------------------

with open('test1.txt', 'w') as file:

    file.write('Wow the With works')

# Random

# Random

# Python has a built-in module that you can use to make random numbers

import random

print(random.random())              # A random float between 0 and 1

print(random.uniform(20, 60))       # A Random float number between a range

print(random.randint(0, 2))         # A random number between the given range (includes both numbers)

print(random.randrange(0,100))      # A random number between the given range (includes both numbers)

my\_list = [1,2,3,4,5,6,7,8,9,0]

print(random.choice(my\_list))       # randomly pick an item from a list

random.shuffle(my\_list)             # shuffle the list

print(my\_list)

print(random.sample(my\_list, k=3))  #Random k size sample of a list

# Command Line Arguments

# Command Arguments

# Python provides a getopt module that helps you parse

# command-line options and arguments

# The Python sys module provides access to any

# command-line arguments via the sys.argv.

# This serves two purposes −

#   sys.argv is the list of command-line arguments.

#   len(sys.argv) is the number of command-line arguments.

# Here sys.argv[0] is the program ie. script name

# Example comsuming commandline arguments

# run this from commandline as

#       py 15\_args.py --help

#       py 15\_args.py --l

#       py 15\_args.py --a [value]

import sys

print(sys.argv)

args\_lower = [item.lower() for item in sys.argv]

print(args\_lower)

my\_list = ['Scott', 'Dave', 'Lisa']

if '--help' in args\_lower:

    print("""

        This is your help system:

        use the following command arguments

        --help (displays this help)

        --l (show list)

        --a (adds value to a list)

    """)

elif '--l' in args\_lower:

    print(my\_list)

else:

    if args\_lower.count('--a'):

        a\_index = args\_lower.index('--a')

        a\_value = sys.argv[a\_index + 1]

        my\_list.append(a\_value)

        print(my\_list)

# Json

# json and csv

# JSON stands for JavaScript Object Notation

# JSON is a lightweight format for storing and transporting data

# JSON is often used when data is sent from a server to a web page

# JSON is "self-describing" and easy to understand

# JSON Syntax Rules

#   • Data is in name/value pairs

#   • Data is separated by commas

#   • Curly braces hold objects

# Square brackets hold arrays

# This is an example of what Json looks like

# {

#     "employees":[

#         {"name":"Scott", "age":21},

#         {"name":"David", "age":34}

#     ]

# }

import json

# Parse a string as json

data = '{"name":"Scott","age":21,"type":"zombie"}'

print(data)

data\_json = json.loads(data)    # use json.loads to convert a string to json

print(data\_json)                # notice this is now json format

print(data\_json['type'])        # grab a value using the key

string\_json = json.dumps(data\_json) # use json.dumps to convert json to a string

print(string\_json)

print('-' \* 50)

# An array of objects in json

names = [

    {

        "name":"Scott",

        "age":21

    },

    {

        "name":"Bob",

        "age": 55

    },

    {

        "name":"Sally",

        "age":17

    }

]

# Write json to a file

with open('file.txt', 'w') as file:

    file.write(json.dumps(names))

# Write nicely formatted json to a file

with open('file.txt', 'w') as file:

    file.write(json.dumps(names, indent=4))

with open('file.txt', 'r') as file:

    all\_names = json.loads(file.read())

print(all\_names[1]['age'])

# CSV

# csv

import csv

# Write csv file using lists

with open('monsters.csv', mode='w', newline ="") as csv\_file:

    writer = csv.writer(csv\_file, delimiter=',', quotechar='"', quoting=csv.QUOTE\_MINIMAL)

    writer.writerow(['Bob', 21, 'zombie'])

    writer.writerow(['Suzette', 34, 'vampire'])

    writer.writerow(['Harry', 42, 'werewolf'])

# Write csv file using dictionaries

with open('monsters2.csv', mode='w', newline ="") as csv\_file:

    fields = ['name', 'age', 'species']

    writer = csv.DictWriter(csv\_file, fieldnames=fields)

    writer.writeheader()

    writer.writerow({

      'name': 'Bob',

      'age': 21,

      'species': 'zombie'

    })

    writer.writerow({

      'name': 'Suzette',

      'age': 34,

      'species': 'vampire'

    })

    writer.writerow({

      'name': 'Harry',

      'age': 42,

      'species': 'werewolf'

    })

# Read csv file into a list of lists

with open('monsters.csv') as csv\_file:

    csv\_reader = csv.reader(csv\_file, delimiter=',')

    my\_list = []

    for idx,row in enumerate(csv\_reader): # csv\_reader can be iterated through but it is not a real list, so we append its values to a list

      print(idx,row)

      my\_list.append(row)

    print(csv\_reader)

    print(my\_list)

    print()

# Read csv file into a list of ordered dictionaries

with open('monsters2.csv', mode='r') as csv\_file:

    csv\_reader = csv.DictReader(csv\_file)

    my\_list.clear()

    for row in csv\_reader: # csv\_reader can be iterated through but it is not a real list, so we append its values to a list

      print(row)

      my\_list.append(row)

    print(csv\_reader)

    print(my\_list)

    print(my\_list[1]['name'])

# Modules

# Modules

# Module are like code libraries, a file containing

# a set of functions you want to include in your application

# Create a module by saving the file with the .py extension

# Import the zz\_sub\_module

import zz\_sub\_module

zz\_sub\_module.greeting('Scott')

# Import the zz\_sub\_module but us an alias

import zz\_sub\_module as mod

print(mod.person\_john['age'])

# Import a specific object from zz\_sub\_module

from  zz\_sub\_module import person\_john

print(person\_john['name'])

# Import everything from the zz\_sub\_module so you don't have

# to reference the module name or alias

from zz\_sub\_module import \*

greeting('Dave')

zz\_sub\_module.py

# this will be a module to import

# declare a function in this module

def greeting(name):

    print(f"Hello, {name}")

# create an object

person\_john = {

    'name': 'John',

    'age': 25,

    'country': 'U.S.A'

}