

Python Primer:

For OO Devs

**Instructor**: Sean Blessing

**Updated:** February 2025

# Overview & Purpose

Provide an overview of the Python programming language in a brief session, from the perspective of a developer who already understands OO programming. This allows an expedited approach, focusing on syntax differences, high level explanation, in class demonstrations, and hands-on labs demonstrating these differences through simple console projects.

# Prerequisites

1. Development experience in another OO language such as Java or C#.
   1. We won’t be going in depth to explain core concepts like control structures, methods, classes, OO concepts in general
   2. We will be doing brief, in class demonstrations, followed up by lab time, if time allows
2. Experience with multiple IDEs such that utilizing a new one is a short learning curve
3. Basic understanding of the Windows Operating System

# Objectives

1. Learn basic console app development in Python.
2. Understand major differences in Python approach vs C based languages.
3. If time allows, introduction to advanced / convenience functions available in Python which simplify some of the things that normally take a lot more code to achieve! Ex: Data Analysis, File Processing, GUI

# Outline

* Python Install
* Python Overview
* Key Differences Overview
* Python / IDE Overview, First Apps, statements, variables, basic control structures (if, while)
* Operators, Math module
* More Control Statements
* Functions, Exceptions
* Debugger
* Classes
* Lists / Dictionaries, looping, finding, processing
* File Processing

# 

# Setup

1. Python Install
   1. Download the latest version [here](https://www.python.org/downloads/). Version 3.13.2 as of February, 2025
   2. Be logged in as an administrator on your laptop
   3. Run the executable
   4. Check the box to add Python to PATH
   5. During install, if you see an option “Disable path length limit”, click it
   6. To confirm install, go to a command prompt and type **python -V** [‘V’ must be uppercase]. This should show you the current installed version.
2. IDE - Most IDEs will have a Python interpreter installed, so you are welcome to use whichever you prefer. The examples in this material use VS Code, but any IDE should work.

# Python Overview / Definition

***From Python.org:*** “Python is an **interpreted**, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with **dynamic typing** and **dynamic binding**, make it very attractive for Rapid Application Development, as well as for use as a **scripting** or glue language to connect existing components together. Python's ***simple, easy to learn syntax*** emphasizes readability and therefore reduces the cost of program maintenance.”

***From Wikipedia:*** “Python is an interpreted high-level **general-purpose programming language.** Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.”

# 

# Python vs Java and C# - [Python.org](https://www.python.org/doc/essays/comparisons/)

* Python programs are generally expected to run slower than Java/C# programs
  + Because of the run-time typing, Python's run time must work harder than Java's.
* Python Programs take much less time to develop

# PIP

* PIP is the package manager in Python
* Similar to NuGet in C#, Maven in Java
* Use it when you need a package that’s not part of core Python

# Sean’s Take

* Python is kind of a jack of all trades language… OO, Functional and Scripting
* That sounds really cool, multi-functional, but if you are used to a traditional OO language it can be tough to get used to
* Python provides a lot of ‘convenience’ functions out-of-the-box… conveniences that we’d otherwise have to write multiple statements to perform the same functionality. See [here for a list of Python built-in functions](https://docs.python.org/3/library/functions.html)
* It is used A TON in Data Science and AI
* Syntax differences are the biggest hurdle for Java, C# programmers looking to skill-up in Python. As I always say, “Repetition Breeds Cognition”!!!

# 

# Biggest Differences (Python vs Java/.Net)

|  |  |  |
| --- | --- | --- |
| No semicolons! | Colons used at start of control structures | “print” vs System.out…, Console.print... |
| No curly braces! | “elif” vs “else if” | “True” vs “true” |
| Indentations mark ‘blocks’ | “not” is synonymous w/ “!” | No ++, -- operator?!?! |
| Dynamic typing, types defined at assignment (think JavaScript) | Can run as script without full class structure | Variable scope is WAY different |
| Types available | var\_name\_convention | Dictionary (Map) usage very different |
| Function definition syntax ‘def’ keyword | Class constructor replaced by \_\_init\_\_() function | IDE difference… lack of auto line breaks after function/ method definition |
| None vs null | ‘self’ vs ‘this’ | No ‘new’ keyword |

* Python source code is ‘interpreted’ into bytecode.
* They are stored in a ‘pychache’ folder, but only once we’re utilizing modules

# A Lot of Stuff is the Same

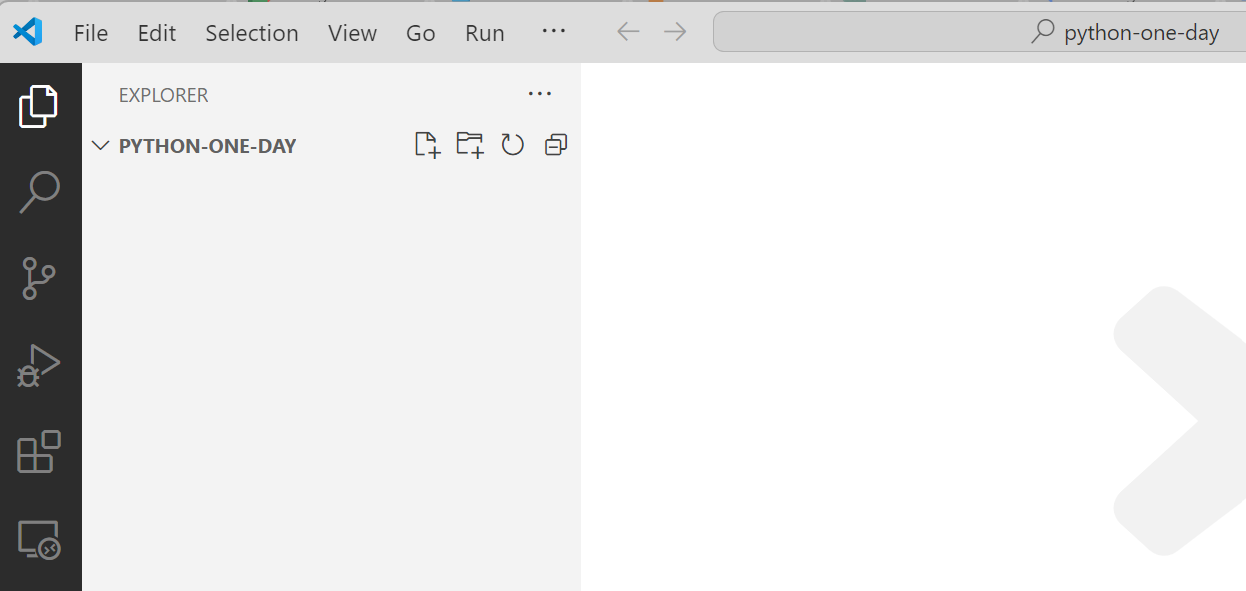
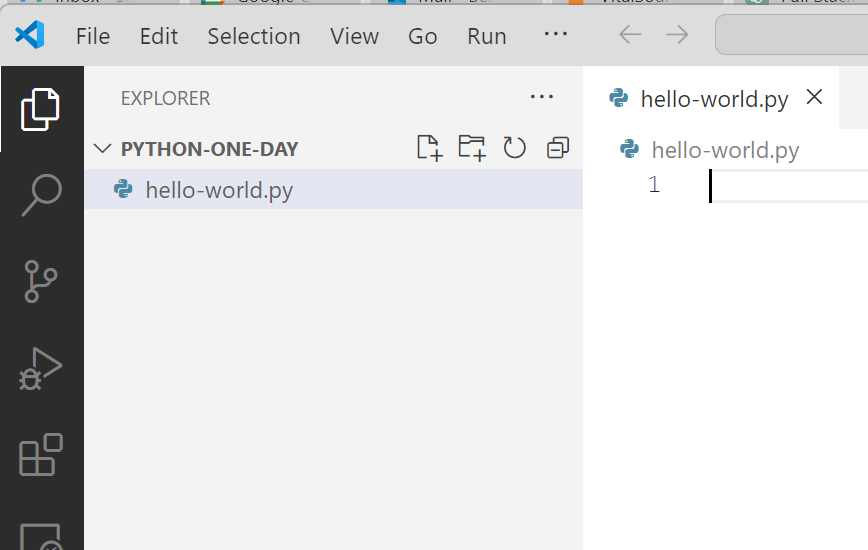
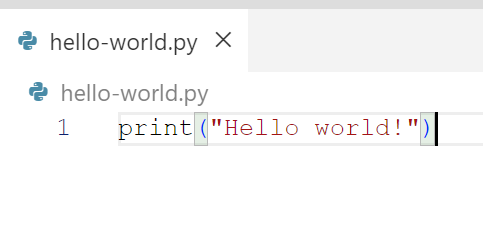
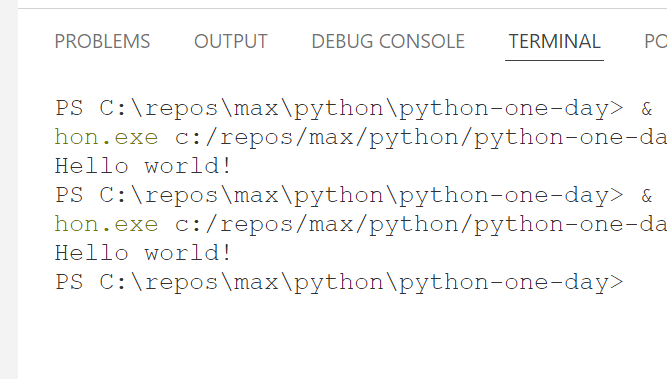
* Top down code execution
* Creating a project
* “if”, “while” statements, “break”, “continue” keywords
* +=, -=, == operators

# Similarities to JavaScript

* Dynamic typing
* Single vs double quotes are interchangeable
* “slice” of lists
  + In both Java and C# there is no explicit ‘slice’, but other methods do something similar [ex: copyOfRange(), substring() in Java]

# Open VS Code

Open a new folder in VS Code, one where we will be writing our Python code.

* 
* Create a new file named hello-world.py  
  
* Add the following line to the file:  
  
  + Run the file:  
    

# Course Approach

|  |  |
| --- | --- |
| Chapter(s) | Concept |
| 1, 2 | Python / IDE Overview, First Apps, statements, variables, basic control structures (if, while) |
| 3 | Operators, Math module |
| 4 | More Control Statements |
| 5 | Functions, Exceptions |
| 6 | Debugger |
| 7 | Classes |
| 8 | Lists / Dictionaries, looping, finding, processing |
| 9 | File Processing |

# 

# Chapter 2 - first applications

* Code completion and syntax error detection
* Statements
  + No semicolon to end the statement!
  + Statements end w/ ‘Enter’ key
* No identifiers… just define your variables and type gets assigned dynamically
* Comments
  + Only single line comments - utilize #
* Basic data types - [w3schools](https://www.w3schools.com/python/python_datatypes.asp)
* Arithmetic operators: +, -, \*, /, %
  + Most data types you are used to are available, just named a little differently
* You can utilize a lot of built in functions, demonstrated in class
* Console interaction:
  + No System.out.println(), System.out.print()
  + No Console.Write(), Console.WriteLine()
  + print(“...” + ”...”): Print whatever you want! You CAN concatenate, but it gets tricky when you introduce variables
  + print(f”....{ variable\_name }...”) -> similar to interpolation
  + Always ends w/ a new line (“\n”).
  + ‘end’ attribute
    - If you want to end a print without a new line:  
      print(“....”, end=””)
  + Print multiple variables/values w/ a separator
    - print(‘a’, ‘b’, ‘c’, sep=”-”) -> a-b-c
  + Get user input: input(“....”)
    - String inside parenthesis will be the prompt
    - Always returns a string
    - Many ways to [convert strings](https://stackabuse.com/convert-strings-to-numbers-and-numbers-to-strings-in-python/)
      * int(...), float(...)
    - Convert numerics to strings - str(...)
    - Get input from arguments passed to program
      * import sys  
        print(sys.argv[0]) -> prints program name  
        print(sys.argv[1]) -> prints 1st argument passed
* If statements

|  |
| --- |
| code = "r" if code == "r":  print("Red") elif code == "b":  print("Blue") else:  print("Other") |

* + Notes:
    - Curly braces replaced w/ single colon
    - “elif” replaces “else if”
* While loop

|  |
| --- |
| choice = "y" while choice != "n":  print("hello")  choice = input("Go again?") |

### Labs

* Grade Converter [2-2]
  + New Concepts Demonstrated:
    - print
    - input
    - while loop
    - if statement
* Rectangle Calculator [2-3]
  + New Concepts Demonstrated:
    - Calculations

# 

# Chapter 3 - computations, Math module, formatting

Built in functions:

* max / min

|  |
| --- |
| max\_nbr = max(5, 7) #7 min\_nbr = min(8, 3) #3 |

* Note: Variable naming convention: snake\_case, or python\_case / snail\_case
* sum

|  |
| --- |
| total = sum([2,3]) #5 total = sum([2,3,5]) #10 |

* Round

|  |
| --- |
| round\_nbr = round(4.567, 2) #4.57 |

* Python [math module](https://docs.python.org/3/library/math.html)

|  |
| --- |
| import math ceil\_nbr = math.ceil(5.5) #6 floor\_nbr = math.floor(5.5) #5 |

* + Note: math module must be imported
* Python [random module](https://docs.python.org/3/library/random.html)

|  |
| --- |
| import random die\_roll = random.randint(1,6) #a random # between 1 and 6, inclusive |

* String formatting - several options!

|  |
| --- |
| price = 20000 price\_currency = "${:,.2f}".format(price) #$20,000.00 grade = .9995 grade\_pct\_1 = format(grade, '%') #99.950000% grade\_pct\_2 = "{:.2%}".format(grade) #99.95% print(f"Format % w/ 2 decimals in string: {grade:.2%}") #99.95% |

### Labs

* Temperature Converter [3-1]
* Travel Time Calculator [3-2]

# Chapter 4 – arrays & more control statements

* Arrays allow us to store a set of values:

|  |
| --- |
| # Arrays  even\_numbers = [2, 4, 6, 8, 10]  odd\_numbers = [1, 3, 7, 9]  # add value at end:  even\_numbers.append(12)  # add value at position (index starts at 0):  odd\_numbers.insert(2, 5)  print(even\_numbers)  print(odd\_numbers) |

* Python doesn’t have a **switch** statement, but it does have a **match** statement:

|  |
| --- |
| command = "list" match command:  case "list":  print("List selected")  case "add":  print("Add selected")  case "edit":  print("Edit selected")  case \_: #default case  print("Unknown Command. Try again.") |

* For loop examples - several variations:

|  |
| --- |
| even\_nbrs = [2, 4, 6, 8, 10] for nbr in even\_nbrs:  print(nbr) # range starts at 0 (default), stops at 4) for i in range(len(even\_nbrs)):  print(f"{i}: {even\_nbrs[i]}")  for i in enumerate(even\_nbrs):  print(i, i[0], i[1])  for idx, value in enumerate(even\_nbrs):  print(f"{idx}: {value}")  # including 'start' (inclusive) and 'stop' (exclusive) for n in range(1, 10):  print(n)   # including 'step' for n in range(1, 10, 2):  print(n) |

* Notes:
  + Square brackets denote a **list** - more on **lists** later.

### Labs

* + 4-1 Table of Powers
  + **4-2 Factorial Calculator**
  + 4-3 Tip Calculator

# Chapter 5 - functions, exceptions

* Python has both functions and methods
  + In a procedural program / script they are called functions
  + In a class they are called methods
* Use the keyword ‘def’ to define a function:

|  |
| --- |
| def add\_function(num1, num2):  return num1 + num2  print(add\_function(5, 3)) |

* Assigning default values:

|  |
| --- |
| def add\_function(num1, num2=2):  return num1 + num2  print(add\_function(5)) |

* Variable args (\*args):

|  |
| --- |
| def add\_function(\*nbrs):  sum = 0  for n in nbrs:  sum += n  return sum  print(add\_function(1, 2)) print(add\_function(2, 4, 6)) print(add\_function(1, 3, 5, 7, 9)) |

* Keyword args

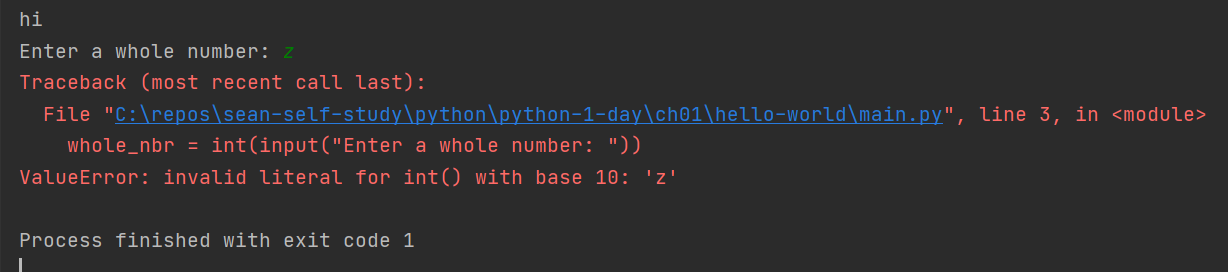
|  |
| --- |
| def calc\_total\_function(price, quantity, handling\_fee):  #(price \* quantity) + handling\_fee = total  return (price\*quantity) + handling\_fee print(calc\_total\_function(20, 2, 3)) #43 print(calc\_total\_function(handling\_fee=5, quantity=7, price=10)) #75 |

* Arbitrary Keyword Arguments(\*\*kwargs) -don’t know how many args will be passed

|  |
| --- |
| def my\_function(\*\*a\_person):  print ("Name is: " + a\_person["l\_name"] + ", " + a\_person["f\_name"])  my\_function(f\_name = "Bob", l\_name = "Marley") |
|  |

* [Access modifiers](https://www.geeksforgeeks.org/access-modifiers-in-python-public-private-and-protected/):
  + Public: The default access, no attribute needed
  + Protected: Single underscore (‘\_’)
  + Private: Double underscore (‘\_\_’)
* Exceptions - treated similarly as we do in other OO languages but the syntax is just a little different.
* Consider the following code and how it could possibly throw an exception:
  + Notes:
    - **input()** accepts user input from the console
    - **int()** converts a string to a number

|  |
| --- |
| whole\_nbr = int(input("Enter a whole number: ")) print(f"You entered: {whole\_nbr}") |

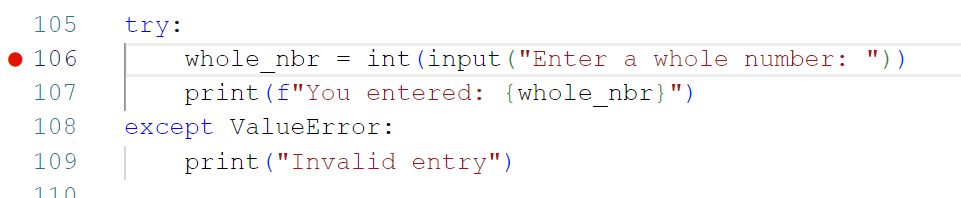
* + If an alphanumeric value is entered, a ValueError is returned:  
    
  + We can wrap the code in a try/except [Not try/catch!]:

|  |
| --- |
| try:  whole\_nbr = int(input("Enter a whole number: "))  print(f"You entered: {whole\_nbr}") except ValueError:  print("Invalid entry") |

### Labs

* 5-1 Dice Roller
* **5-3 Guessing Game**

# Chapter 6 - debugger

* Set a breakpoint by clicking just to the left of the line number in your editor. A red dot will appear.  
  
* The integrated debugger is accessed under the Run -> Start Debuging... menu item.
* Typical debugger options are available (step over, step into, etc.) from the Run menu option.

# Chapter 7 - classes, modules

* With Python version 3.7 the **@dataclass** decorator was introduced, which directs Python to generate the **\_\_init\_\_()** method. More on this later.
* Here’s an example of a **Person** class, defined in Python v3.7 or later::

|  |
| --- |
| from dataclasses import dataclass  @dataclass class Person:  id: int  first\_name: str  last\_name: str  email: str  phone: str |

* + ‘class’ declaration - assumed public
  + Class names start with a capital letter.
  + Class names DO NOT have to match the filename.
  + Property names defined as snake\_case w/ types
  + No declaration of getter, setters, nor constructor
* A **Person** class defined in Python v6 or earlier:

|  |
| --- |
| class Person:  def \_\_init\_\_(self, id = 0.0, first\_name = "", last\_name = "",  email = "", phone = ""):  self.id = id  self.first\_name = first\_name  self.last\_name = last\_name  self.email = email  self.phone = phone |

* + Properties defined inside the **\_\_init\_\_()** method
  + The **self** keyword is used to set properties, similar to **this** in other languages
* A person\_manager.py file importing Person, creating an instance, displaying properties, and changing a property:

|  |
| --- |
| from person import Person  person = Person(1, "Marty", "McFly", "marty@b2f.com", "282-333-1234") print(f"Person Info:") print(f"id: {person.id}") print(f"name: {person.first\_name} {person.last\_name}") print(f"email: {person.email}") print(f"phone: {person.phone}") person.phone = "222-333-1234" print(f"phone: {person.phone}") |

* + Notes:
    - Constructing a new instance does NOT require the new keyword
    - Properties are accessed using dot notation (**person.phone**)
* Methods can be defined in a class by using the **def** keyword:

|  |
| --- |
| def get\_person\_details(self):  details = f"id: {self.id}, first\_name: {self.first\_name}, last\_name: {self.last\_name}, "  details += f"email: {self.email}, phone: {self.phone}"  return details |

* **details** **f string** is split across two lines. Since Python utilizes end of line characters to end statements, we split it into two statements.

### Labs

* + 7-1 Contact List
    - Contact
      * first\_name
      * last\_name
      * email
      * Phone
      * print\_contact()
    - Prompt user for 4 values of a contact
    - Create an instance of Contact
    - Print the contact info

# Chapter 8 – strings as lists, dictionaries, etc.

Every language has various types to manage “lists” of things. Here we will talk about Ptyhon’s list and dictionary types.

* A Python **list** is similar to an **array** in other languages, yet is more dynamic. It is similar to an **ArrayList** in Java or C#. Here are some examples:

|  |
| --- |
| # A string: chars stored in order bob\_marley\_name = "Robert Nesta Marley"  # A list: ordered and changeable, denoted by square brackets songs = ["Jamming", "Three Little Birds", "No Woman No Cry"]  # A tuple: ordered and NOT changeable, denoted by parenthesis # Note: this tuple is not complete. Bob Marley had a lot of kids! children = ("Sharon", "David 'Ziggy'", "Stephen", "Robert 'Robbie'", "Julian", "Ky-Mani", "Damian") |

* Processing a string - **bob\_marley\_name**:

|  |
| --- |
| #processing a string - bob\_marley\_name bob\_marley\_name = "Robert Nesta Marley" print('bob\_marley\_name:', bob\_marley\_name) print('length of bob\_marley\_name', len(bob\_marley\_name)) #get portions of the bob\_marley\_name by index and slicing print('first char:', bob\_marley\_name[0]) #get first name of bob\_marley\_name # where's the first space? idx\_1 = bob\_marley\_name.index(" ") f\_bob\_marley\_name = bob\_marley\_name[0:idx\_1] print(f"First name: {f\_bob\_marley\_name}") idx\_2 = bob\_marley\_name.index(" ", idx\_1+1) m\_bob\_marley\_name = bob\_marley\_name[idx\_1+1: idx\_2] print(f"Middle name: {m\_bob\_marley\_name}") l\_bob\_marley\_name = bob\_marley\_name[idx\_2+1:len(bob\_marley\_name)] print(f"Last name: {l\_bob\_marley\_name}") for str in bob\_marley\_name:  print(str, end=" ") |

* Output:

|  |
| --- |
| bob\_marley\_name: Robert Nesta Marley length of bob\_marley\_name 19 first char: R First name: Robert Middle name: Nesta Last name: Marley R o b e r t N e s t a M a r l e y |

* Processing a list:

|  |
| --- |
| # process the songs in a list # use square brackets and slicing songs = ["Jamming", "Three Little Birds", "No Woman No Cry"] print(f"songs: {songs}") print(f"song 1: {songs[0]}") print(f"last song songs[2]: {songs[2]}") print(f"last song songs[-1]: {songs[-1]}") print(f"last 2 songs songs[1:3]: {songs[1:3]}") #append a song to the end songs.append('temp song') print(f"temp song added to end: {songs}") #change a song/item songs[3] = "Bobs so cool - not a real song" print(f"song 3 changed: {songs}") # remove the last song and return it rem\_song = songs.pop() print(f"removed song: {rem\_song}") print(f"songs: {songs}") |

* Output:

|  |
| --- |
| songs: ['Jamming', 'Three Little Birds', 'No Woman No Cry'] song 1: Jamming last song songs[2]: No Woman No Cry last song songs[-1]: No Woman No Cry last 2 songs songs[1:3]: ['Three Little Birds', 'No Woman No Cry'] temp song added to end: ['Jamming', 'Three Little Birds', 'No Woman No Cry', 'temp song'] song 3 changed: ['Jamming', 'Three Little Birds', 'No Woman No Cry', 'Bobs so cool - not a real song'] removed song: Bobs so cool - not a real song songs: ['Jamming', 'Three Little Birds', 'No Woman No Cry'] |

* Range:

|  |
| --- |
| #range function print("\nRanges!") print(list(range(10))) print(list(range(5, 17))) print(list(range(0, 20, 2))) print(list(range(0, 20, 3))) print(list(range(0, -10, -1))) |

* Output:

|  |
| --- |
| Ranges! [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] [0, 2, 4, 6, 8, 10, 12, 14, 16, 18] [0, 3, 6, 9, 12, 15, 18] [0, -1, -2, -3, -4, -5, -6, -7, -8, -9] |

* ‘For .. in’:

|  |
| --- |
| # for .. in print("\nfor .. in") for song in songs:  print(song) |

* Output:

|  |
| --- |
| Ranges! [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] [0, 2, 4, 6, 8, 10, 12, 14, 16, 18] [0, 3, 6, 9, 12, 15, 18] [0, -1, -2, -3, -4, -5, -6, -7, -8, -9] |

* ‘for’ using index:

|  |
| --- |
| # for w/ index print("\nfor w/ index") for i in range(0, len(songs)):  print(f"song [{i}]: {songs[i]}") |

* Output:

|  |
| --- |
| for w/ index song [0]: Jamming song [1]: Three Little Birds song [2]: No Woman No Cry |

* ‘in’ keyword - see if something is in a list:

|  |
| --- |
| # in keyword - text for existence in a list: print("'in' keyword - is 2 in the list?") numbers\_list = [1, 2, 1, 3, 1, 4, 1, 5] if 2 in numbers\_list:  print("yep, it's in there!") |

* Output:

|  |
| --- |
| 'in' keyword - is 2 in the list? yep, it's in there! |

* The ‘\*’ operator does more than math when used on strings and lists:

|  |
| --- |
| # The \* operator - more than math multiplier print("\n\* operator") str = "a" \* 10 print(str) numbers\_list = [1, 2, 3] num\_list = numbers\_list \* 3 print(f"num\_list: {num\_list}") |

* Output:

|  |
| --- |
| \* operator aaaaaaaaaa num\_list: [1, 2, 3, 1, 2, 3, 1, 2, 3] |

* Here are some [built-in methods for lists](https://www.tutorialspoint.com/built-in-list-functions-and-methods-in-python)
  + A few more: clear(), copy()
* Use random() to get a random item from list:

|  |
| --- |
| # use random() to get a random item from a list: print("\nrandom # from list") import random numbers\_list = [1, 50, 23, 74, 11] print(f"1st choice: {random.choice(numbers\_list)}") print(f"2nd choice: {random.choice(numbers\_list)}") |

* Output:

|  |
| --- |
| random # from list 1st choice: 23 2nd choice: 11 |

* Dictionaries

|  |
| --- |
| # dictionaries # dictionary key/value can contain any data type # can access by get method or square brackets print("\ndictionaries") spanish\_english = {  "uno": "one",  "dos": "two",  "tres": "three",  1: "one" } print(spanish\_english) # get value by key print(spanish\_english.get("dos")) print(spanish\_english[1]) # add elements to a dictionary using square brackets (no add function) spanish\_english["quatro"] = "four" print(spanish\_english) |

* Output:

|  |
| --- |
| dictionaries {'uno': 'one', 'dos': 'two', 'tres': 'three', 1: 'one'} two one {'uno': 'one', 'dos': 'two', 'tres': 'three', 1: 'one', 'quatro': 'four'} |

* Iterate through dictionary keys using for...in

|  |
| --- |
| #iterate over dictionary keys using for .. in print("\nIterate over dictionary w/ for .. in") for key in spanish\_english:  print(f"{key} means {spanish\_english[key]}") |

* Output:

|  |
| --- |
| Iterate over dictionary w/ for .. in uno means one dos means two tres means three 1 means one quatro means four |

* .values() returns all values:

|  |
| --- |
| #.values contains all values print("\n.values() to return all key-value pairs") for value in spanish\_english.values():  print(f"{value}", end=" ") |

* Output:

|  |
| --- |
| .values() to return all key-value pairs one two three one four |

* Get both keys and values using .items():

|  |
| --- |
| #get both keys and values using .items() print(".items() returns both key and value") for k, v in spanish\_english.items():  print(f"k, v: {k}, {v}") |

* Output:

|  |
| --- |
| .values() to return all key-value pairs one two three one four .items() returns both key and value k, v: uno, one k, v: dos, two k, v: tres, three k, v: 1, one k, v: quatro, four |

* Check to see if a key exists in dictionary:

|  |
| --- |
| #does key exist in dictionary? print("check key existence...") if "uno" in spanish\_english:  print("Yes, 'uno' is in dictionary") |

* Output:

|  |
| --- |
| check key existence... Yes, 'uno' is in dictionary |

* Use pop() to remove last item or pop(key) to remove specific item:

|  |
| --- |
| #use pop(key) to remove item or popitem() to remove last item #pop returns removed value, popitem() returns entire item (k,v) print("\npop and popitem to remove items") print(spanish\_english) item\_1 = spanish\_english.pop(1) print(f"item\_1={item\_1}") print(spanish\_english) item\_2 = spanish\_english.popitem() print(f"item\_2={item\_2}") print(spanish\_english) |

* Output:

|  |
| --- |
| pop and popitem to remove items {'uno': 'one', 'dos': 'two', 'tres': 'three', 1: 'one', 'quatro': 'four'} item\_1=one {'uno': 'one', 'dos': 'two', 'tres': 'three', 'quatro': 'four'} item\_2=('quatro', 'four') {'uno': 'one', 'dos': 'two', 'tres': 'three'} |

# 

# Chapter 9 - file processing

* Python provides some very simple functions to process files
* Much less overhead than in Java SDK, for sure!

## Text File Example

* Given a text file hobbies.txt:

|  |
| --- |
| music running biking python coding |

* + File methods:
    - read() - reads whole file as a single long string
    - readline() - reads one line of file into string
    - readlines() - reads file into list of strings
* We can process this file with the following code:

|  |
| --- |
| print("File Processing") # read() print("read() hobbies.txt") with open('hobbies.txt') as hobbies\_file:  contents = hobbies\_file.read() print(f"read() : {contents}")  # readline() print("readline() hobbies.txt") with open('hobbies.txt') as hobbies\_file:  contents = hobbies\_file.readline() print(f"readline() : {contents}")  # readlines() print("readlines() hobbies.txt") with open('hobbies.txt') as hobbies\_file:  contents = hobbies\_file.readlines() print(f"readlines() : {contents}") |

* Output:

|  |
| --- |
| File Processing read() hobbies.txt read() : music running biking python coding readline() hobbies.txt readline() : music  readlines() hobbies.txt readlines() : ['music\n', 'running\n', 'biking\n', 'python coding'] |

* Writing to a file:

|  |
| --- |
| #writing to a file #define list of olympic athletes print("\nWrite to the olympic-athlete file...") olympic\_athletes = ["Simone Biles", "Michael Phelps", "Michael Johnson", "Chloe Kim"] #define a file to write to (without the 'with' keyword) olympic\_file = open("olympic-athletes.txt", "w") #loop through list and write to file for athlete in olympic\_athletes:  olympic\_file.write(f"{athlete}\n") #close file olympic\_file.close() print("Done. Check the file.") |

* Output:

|  |
| --- |
| Write to the olympic-athlete file... Done. Check the file. |

* Check VS Code for the **olympic-athletes.txt** file.

# 

## Processing CSV Files

Python provides the **csv** module to make processing CSV files easy.

### Writing to a CSV

Let’s start with a Movie class, create some movies, and put them in a list:

* The Movie class:

|  |
| --- |
| from dataclasses import dataclass  @dataclass class Movie:  id: int  title: str  year: int  rating: str  director: str   def get\_movie\_details(self):  details = f"id: {self.id}, title: {self.title}, year: {self.year}, "  details += f"rating: {self.rating}, director: {self.director}"  return details |

* The code to use the movie class, create some movies, and write to CSV:

|  |
| --- |
| #processing CSV Files print("CSVs") print("Write to a movies.csv file") #define some movies movie\_1 = Movie(1, "Star Wars Episode IV: A New Hope", 1977, "PG", "George Lucas") movie\_2 = Movie(2, "Coco", 2017, "PG", "Lee Unkrich, Adrian Molina") movie\_3 = Movie(3, "Black Panther", 2018, "PG-13", "Ryan Coogler") #put them in a list movies = [movie\_1, movie\_2, movie\_3] #import csv and use it to write the movies to a file using with keyword import csv with open("movies.csv", "w", newline="") as movie\_file:  writer = csv.writer(movie\_file)  for movie in movies:  writer.writerow([movie.id, movie.title, movie.year, movie.rating, movie.director]) print("done") |

* Output

|  |
| --- |
| CSVs Write to a movies.csv file done |

* Reading a CSV

|  |
| --- |
| # read the csv  print("\nRead the movie csv and print to console...") with open("movies.csv", newline="") as movie\_file:  reader = csv.reader(movie\_file)  for movie in reader:  print(f"Movie: {movie[1]} ({movie[2]}), rated {movie[3]}, directed by {movie[4]}") print("done") |

* Output

|  |
| --- |
| Read the movie csv and print to console... Movie: Star Wars Episode IV: A New Hope (1977), rated PG, directed by George Lucas Movie: Coco (2017), rated PG, directed by Lee Unkrich, Adrian Molina Movie: Black Panther (2018), rated PG-13, directed by Ryan Coogler done |

* Writing from a List of Lists rather than a List of Objects
  + This is a little simpler with the csv library, so it depends on the situation:

|  |
| --- |
| # writing csv from a list of lists print("\nWrite a movies csv from a list of lists") movies\_list = [["Star Wars Episode IV: A New Hope", 1976],  ["Sixteen Candles", 1984],  ["Rogue One", 2016],  ["Happy Gilmore", 1996],  ["Wedding Crashers", 2005]] # write list to a csv with open("movies-list.csv", "w", newline="") as movie\_file:  writer = csv.writer(movie\_file)  writer.writerows(movies\_list)  #p.219 csv.reader() with open("movies-list.csv", newline="") as movie\_file:  reader = csv.reader(movie\_file)  for movie in reader:  print(f"{movie[0]} ({movie[1]})") print("done") |

* Output

|  |
| --- |
| Write a movies csv from a list of lists Star Wars Episode IV: A New Hope (1976) Sixteen Candles (1984) Rogue One (2016) Happy Gilmore (1996) Wedding Crashers (2005) done |

# Chapter 10 - date and time

* Python provides several modules to process time, datetime, etc.
* [W3schools datetime](https://www.w3schools.com/python/python_datetime.asp)
* Basic Date Stuff::

|  |
| --- |
| # basic date stuff print("Basic date definition...") date\_today = date.today() print(f"date\_today: {date\_today}") datetime\_today = datetime.today() print(f"date\_today: {datetime\_today}") datetime\_now = datetime.now() print(f"date\_now: {datetime\_now}") time\_now = datetime\_now.time() print(f"time\_now: {time\_now}")  christmas = date(2024,12,25) print(f"christmas: {christmas}") time\_eleven = time(11,11,11) print(f"time\_eleven: {time\_eleven}") |

* Output

|  |
| --- |
| Basic date definition... date\_today: 2024-05-16 date\_today: 2024-05-16 11:30:17.286177 date\_now: 2024-05-16 11:30:17.286176 time\_now: 11:30:17.286176 christmas: 2024-12-25 time\_eleven: 11:11:11 |

* Datetime parsing strings

|  |
| --- |
| #p.307 datetime parsing strings print("\nParsing datetime strings:") christmas\_date\_str = "12/25/24" halloween\_date\_str = "2024-10-31" appointment\_date\_time\_str = "2024-05-09 08:00"  christmas\_date = datetime.strptime(christmas\_date\_str, '%m/%d/%y') print(f"christmas\_date: {christmas\_date}") halloween\_date = datetime.strptime(halloween\_date\_str, '%Y-%m-%d') print(f"halloween\_date: {halloween\_date}") appointment\_date = datetime.strptime(appointment\_date\_time\_str, '%Y-%m-%d %H:%M') print(f"appointment\_date: {appointment\_date}") |

* Output

|  |
| --- |
| Parsing datetime strings: christmas\_date: 2024-12-25 00:00:00 halloween\_date: 2024-10-31 00:00:00 appointment\_date: 2024-05-09 08:00:00 |

* Formatting Dates

|  |
| --- |
| # #p.309 - formatting dates print("\nFormatting dates:") christmas\_date\_formatted = christmas\_date.strftime("%m/%d/%y") print(f"christmas\_date\_formatted: {christmas\_date\_formatted}") halloween\_date\_formatted = halloween\_date.strftime("%Y-%m-%d") print(f"halloween\_date\_formatted: {halloween\_date\_formatted}") appointment\_date\_formatted = appointment\_date.strftime("%Y-%m-%d %H:%M") print(f"appointment\_date\_formatted: {appointment\_date\_formatted}")  # another way to format a date... halloween\_datetime = datetime(1988, 10, 31, 22, 48) print(f"halloween\_datetime: {halloween\_datetime}") halloween\_date\_formatted = f"{halloween\_datetime:%Y-%m-%d}" print(f"halloween\_datetime\_formatted(v2): {halloween\_date\_formatted}") |

* Output

|  |
| --- |
| Formatting dates: christmas\_date\_formatted: 12/25/24 halloween\_date\_formatted: 2024-10-31 appointment\_date\_formatted: 2024-05-09 08:00 halloween\_datetime: 1988-10-31 22:48:00 halloween\_datetime\_formatted(v2): 1988-10-31 |

* Comparing Dates - deltas

|  |
| --- |
| # spans of time from datetime import timedelta print("--- timedeltas ---") three\_weeks = timedelta(weeks=3) print(f"today = {date\_today}") print(f"Three weeks from today: {date\_today + three\_weeks}") print(f"Three weeks ago today: {date\_today - three\_weeks}") time\_span\_until\_christmas = christmas - date\_today print(f"Time Span until christmas: {time\_span\_until\_christmas}") print(f"Days until christmas: {time\_span\_until\_christmas.days}") |

* Output

|  |
| --- |
| --- timedeltas --- today = 2024-05-16 Three weeks from today: 2024-06-06 Three weeks ago today: 2024-04-25 Time Span until christmas: 223 days, 0:00:00 Days until christmas: 223 |

# 

# Chapter 11 - JSON module

* JSON is similar to a dictionary
* Import json module
* **json.load**: convert JSON to Python dictionary
* **json.dump**: convert Python to JSON
* VS Code: format JSON to readable format - **right-click** -> **Format Document**
* Load Example:
  + JSON file - people.json

|  |
| --- |
| {  "people": [  {  "name": "Mickey Mouse",  "email": "mickey@disney.com"  },  {  "name": "Stitch",  "email": "stitch@disney.com"  },  {  "name": "Tiana",  "email": "tiana@disney.com"  }  ] } |

* + Python code utilizing **people.json**:

|  |
| --- |
| import json  #working with json - load print("working with json") print("read people json and print each person...") with open('people.json') as json\_file:  data = json.load(json\_file)  print(f"type returned from json.load: {type(data)}")  for person in data['people']:  print(f"Person: name={person['name']}, email={person.get('email')}") |

* + Output:

|  |
| --- |
| working with json read people json and print each person... type returned from json.load: <class 'dict'> Person: name=Mickey Mouse, email=mickey@disney.com Person: name=Stitch, email=stitch@disney.com Person: name=Tiana, email=tiana@disney.com |

* Dump example:
  + Python file:

|  |
| --- |
| #working with json - dump print("\njson.dump") data = {} data['actors'] = [] data['actors'].append({  'name': 'Mark Hamill', 'email': 'luke@starwars.com' }) data['actors'].append({  'name': 'Dwayne Johnson', 'email': 'rock@therock.com' })  with open('actors.json', "w") as actor\_file:  json.dump(data, actor\_file)  with open('actors-formatted.json', "w") as actor\_file:  json.dump(data, actor\_file, indent=4) print("done - check files") |

* + Running this file produces 2 actors json files:
    - actors.json - data all on one line

|  |
| --- |
| {"actors": [{"name": "Mark Hamill", "email": "luke@starwars.com"}, {"name": "Dwayne Johnson", "email": "rock@therock.com"}]} |

* + - Actors-formatted.json - formatted as readable json

|  |
| --- |
| {  "actors": [  {  "name": "Mark Hamill",  "email": "luke@starwars.com"  },  {  "name": "Dwayne Johnson",  "email": "rock@therock.com"  }  ] } |

# Jupyter Notebooks and Use in Data Science

# Other Stuff

* [Requests module](https://docs.python-requests.org/en/master/)
* [Jupyter Notebooks](https://jupyter.org/)

page break