Python – Core

Learn to Code with Python
With Notes from the Python Nano Degree
Program

Pip
Python Interpreter

- pip install <package name>
- requirements.txt
  - include lies where each line contain a filename and version
    - version is optional
    - Example form requirements.txt   beautifulsoup4==4.5.1
  - Advantage – If the libraries change dramatically the version being used worked with code current
  - pip install -r requirements.txt   // Install all packages with the current version
  - Protip : pip freeze > requirement.txt
- Python Interpreter
  - Define a function and get three . (…) which is a continuation line
  - can use up and down arrows to go through the commands
  - IPython
    - tab completion
    - ? for details about an object – ex. len?
    - ! to execute system shell commands
    - syntax highlighting

Help Function

- help show the documentation
- example help(len) or help(“len”)
- example help(str)   // displays information about the string object
- example help(“Hello”.replace)   // Shows the replace method
- example([1].extend)
- Most Important Note : Python is case sensitive
- There are four datatypes in Python int, bool, string, float

Mac OS – Using the Terminal
Mac OS – Installing Python
Mac OS- Install Visual Studio Code and Python Plugins
VSCode Interface and Shortcuts
Interactive Prompt

- Shift Command P → Brings up the command pallet (a set of functions to call)
  - Ex (A command in the pallete) select “Shell Command Install ‘code’ command in PATH
    - Allowsyou to open any python file in Visual Studio from the terminal
  - Ex python: select interpreter
  - On the command line code <filename> will open the file
- .vscode → Found in the top level directory and is a json file
  - settings.json & launch.son
- Ctrl Option N will run the program
- Fuzzy Search Command Plus P
- Command : python3
- REPL → (R)ead (E)valuate (P)rint (L)oop
- Shift Option Down will copy the line
- Command + p will bring up a window with files in your project

Basic Input/Output

- Example
  - name = input(“Enter your name”)
    - parameter – Prompt for question
    - return a string
    - An string can used directly, but int, float, bool must be convert from a string
  - print(“Hello there, {}!”.format(name.title()))
- input always return a string

Object String Function

- Everything is an Object
- String – An immutable sequence of text characters
  - Literal : A piece of syntax that creates an object
- Quote Type Use “ if the string has a ‘ and ‘ if it has double quotes as part of the String
- Empty String has no characters
- Need to insert a double quote inside a stirng
  - can use single
  - use a backslahs ex. \”
- Triple quotes are multiline strings
  - Example “””my name is
    - Chuck
    - “””
  - Used a documentation strings
- Functions
  - Two Type of functions : built-in and custom

The Print Function 1 & 2 & 3

- Print(“3” + “4”) outputs 34
- The print contains *values: object as a parameter which means it can print out different objects that are comma separated
  - *values:object is the parameter declaration
- Example print(“A”, “B”), print(5 +3. 2 – 9) will print “8 -7”
- Default arguments
  - print(*object, sep=’ ‘, end=’\n’, file=sys.stdout, flush=False)
  - Sep=’ ‘ is the reason why we see a single space between each object printed out
  - print(“ABC”, “DEF”, “XYZ”, sep=”!”) will produce ABC!DEF!XYZ!
- Name arguments are useful since you can use them after variable arguments of the same type
  - A keyboard parameter is a name for the parameter that can be used in the calling function
    - ex. print(“ABC”, “DEF”, “XYZ”, sep=”!”) will produce ABC!DEF!XYZ!
- Example – print(“Mohammed has {} ballons”.format(27))
- Example – print(“Does you {} {}?”.format(animal, action)
  - prints Does your dog bite assuming animal = “dog” and action=”bite”

The format String

- Examples
  - Note I believe all these example below need the .format with the correct parameters
  - First, thou shalt count to {0}" # References first positional argument
    - "Bring me a {}"   # Implicitly references the first positional argument
    - "From {} to {}"   # Same as "From {0} to {1}"
    - "My quest is {name}"   # References keyword argument 'name'
    - "Weight in tons {0.weight}"   # 'weight' attribute of first positional arg
    - "Units destroyed: {players[0]}"   # First element of keyword argument 'players'.
  - Examples
    - "Harold's a clever {0!s}"   # Calls str() on the argument first
    - "Bring out the holy {name!r}"   # Calls repr() on the argument first
    - "More {!a}"   # Calls ascii() on the argument first

The Template Strings

- >>> from string import Template
- >>> s = Template('$who likes $what')
- >>> s.substitute(who='tim', what='kung pao')
  - 'tim likes kung pao'
- >>> d = dict(who='tim')
- >>> Template('Give $who $100').substitute(d)
  - Traceback (most recent call last):
    - ValueError: Invalid placeholder in string: line 1, col 11
- >>> Template('$who likes $what').substitute(d)
  - Traceback (most recent call last):
  - KeyError: 'what'
- >>> Template('$who likes $what').safe_substitute(d)
  - 'tim likes $what'

Comments

- If the first character is a # then it is a comment
- Anything after the # is a comment
  - Print(1 + 1) # Adds together 1 + 1

Mathematical Functions

- An expression is a python line that is evaluated
- Operators : + (for numbers and strings) , -, *, * (for numbers and string), **, /, //, %.
  - other operators +=, -=, *=, /=
  - Print (“lolo” * 3) produces lolololololo
  - // (Integer Division) – Divides and rounds down to the nearest integer
    - example print(-7 // 2) would be -4
  - ^ – Performs bitwise xor
  - String + will combine string
  - String * will repeat the string ^*3 // would output ^^^
- Different different datatypes
  - Print (10 + 3.8) // Converts to the specific datatype (int become float) answer is 13.8
  -

Division, Floor Division and the Modulo Operator

- print (15 / 3) produce 5.0
- print(14/3) produces 4.6666666667
- Floor division :
  - print(14 // 3) produces 4
  - print(-14 // 3) produces -5 (rounds down to the nearest integer)
- Modulus Operator : print(14 % 3) displays 2

The Boolean Data Type, The Equality Operator (==) and Inequality
Boolean Mathematical Operators
Type

- Boolean is another data type in python
- Operators ==, !=, <, <=, >, >=, and, or, not
- print (8.3 == 8.3) // True
- print (5 == 5.0) // True
  - The 5 is converted to 5.0
- Boolean Mathematical Operators
  - print(5 < 8 <= 10) // Returns true
- Type – Built in Type to return the Type
  - print(type(5)) // Returns <class ‘int’>

Type Conversion with int, float and conversion

- Conversion functions int, float, str
- Example
  - print (int(6.1)) // prints 6
    - No rounding , takes the floor
  - print(int(“3”)) // prints 3
  - print(float(“5”)) // prints 5.0
  - print(str(5.35)) // print 5.35
- Where the conversions will be automatically performed
  - first_name, last_name, *details = employee would produce
- Example print(type(4.3)) would produce <class ‘float’>
- Example 453 would be considered a float
- Floats are approximation for .1 (actually slightly more than .1)
  - print(.1 + .1 + .1) would produce .3000000000001
  - print(.1 + .1 + .1 == .3) would produce false

Variables
Multiple Variable Assignment

- Use a variable name without be initialized will force a “NameError”
- Rules
  - No Spaces allowed
  - The first character must be a letter or underscore
  - Only letters, number and underscores are permitted after the first character
  - Case sensitive
  - Pythonic way : Use lower case letter and underscore only.
- Multiple Variable Assignments
  - Example
    - a = b = 5
    - b = 10
    - then a = 5 and b = 10
  - Example a,b = 5,10 // Using the datatype Tuples
  - Example
    - x, y, z = 2, 3, 5 // produces x=2, y=3, z=5

Augmented Assignment Operator
User Input with Input Function
NameError, ValueError, TypeError, SyntaxError

- Example
  - a += 2   += is the Augmented Assignment Operator
  - word += car
- User Input with Input Function
  - Example
    - InputText = input(“Enter Some Text “)
      - Add a space at the end so the user can see where the cursor Starts or Ends
      - InputText is the text the user has entered
- NameError, ValueError, TypeError
  - NameError – The interpreter cannot recognize a name that is being referred in the program
    - Ex : The name of variable that is not declared.
  - ValueError – Raised when the function receives an argument that has the right type, but a inappropriate value
    - Ex. A string being passed to the int function that container letters
  - TypeError – Raised when an operation s applied to an inappropriate value
    - Adding a string and integer
  - SyntaxError – Raised code cannot be evaluated
- In operations python will evaluate the left side of the operand to the actinos needed. Print(3 + “5”) and print(“5” + 3) will produce a different error

Other Errors

- ZeroDivisionError
- SyntaxError: unexpected EOF while parsing
  - This message is often produced when you have accidentally left out something
- "TypeError: len() takes exactly one argument (0 given)"

Intro to Functions
Parameters and Arguments
Positional Arguments and Keyword Arguments

- pass → A reserved keyword that is a placeholder for a block
- Parameters and Arguments
  - If you call a function and forget an argument then a TypeError will be raised
- Positional Arguments and Keyword Arguments
  - Example
    - def add(a,b,c):
      - print(“The sum of”, a “and”, b, “is”, a + b + c)
    - add(4,6)
    - add(a=4, b=6, c=3)   // Example of Keyword Arguments
    - add(b=6, a=4, c=3)   // Example of Keyword Arguments
    - add(5, b = 10, c= 15);
    - Add(b=10, 5, c=10)   // Would produce an error “non keyword arg after keyword arg”)
  -

Return Values
Default Arguments for Function Parameters
None Type

- None is a special object that represents nothingness
  - Example print the return value of a function when nothing is returned
- Default Functions Argument Values
  - Example
    - def add(a = 0, b=0):
      - return(a,b)
    - add(10) // 10 will be assigned to1 0 and b will be assigned to 0
    - add() // The value will be 0
  - Optional argument must defined at the end of the parameter list
- None Type
  - Example a = None
  - Class is None Type
  - If a function does not return anything it returns none.

Function Annotation

- Static typing the type of the value cannot change.
- Additional information (metadata) about the function
- Example
  - def wordMultiplier(word:str, time:int) → str: // :str, :int is an example of metadata and so is the str after “str”
    - return word * times
  - Not doing any type checking. If you pass in word = 10 then the return value would be 50
  - Consider it documentation
- Commenting Functions
  - Documentation Strings are a type of comment to explain a function
  - Surrounded with “””

Strings: Length Contention and Immutability

- print(len(“Python”)) would produce 6
  - len can be used for different objects
  - print(len(4)) will give you a Type Error Exception
- String are immutable
  - If two String are concatenated then a new String is created
- Example: print(“---” * 30)   // Will repeat the string 30 times
- Immutability – When a change is made to an immutable string then a new object is crated
- Number, Floats, String, Boolean are Immutable

String Indexing with Positive Values
String Indexing with Negative Values

- All Array are 0 indexed
- Example a[0] = 5 // Get the error “Object does not support item assignment
- Example a has 10 characters, but a[100] produces an “Index out of range” (IndexError)
- String Indexing with Negative Values
  - Extract from the end of the string
  - The last character is -1 and the second to last character is -2

String slicing 1

- Slicing → A form of indexing that returns a selection of characters or elements in a list
  - lower index is inclusive
  - upper index is exclusive
- Example
  - address = “Attractive Street, Beverly Hills, CA 90210
  - [start_index:end_index]
    - start_index is inclusive
    - end_index is exclusive
  - Example address[0:3] produces Att
  - Example address[10:100] produces “Street, Beverly Hills, CA 90210”
  - Example address[34:-6])   Would produce CA
  - Example address[-8:-6] Would produce CA
  - Example address[-8:36] Would produce CA
  - Example address[5:]   ctiveStreet, Beverly Hills, CA 90210
  - Example address[:10]   Attractive
  - Example address[:]   ”Attractive Street, Beverly Hills, CA 90210”
  - “commando”[3:7]   and

Slicing By Steps

- Alphabet =”abcdefghijklmnopqrstuvwxyz”
  - print(alphabet[0:10:2]) produces acegi
  - print(alphabet[0:10:2]) produces adgjmpsvy
- Reversing the String : print(alphabet[::-1]

Escape Characters

- Escape Characters: \n \t \” \’
- Raw Strings – Just a collection of character with out any interpreted characters
  - Used for filenames since they have \
  - file_name = “C:\news\travel”
    - Use r”C:\news\travel”
    - print(file_name) will produce C:\news\travel
- Break up your code on multiple lines use the \ at the end of line

The in and not in Operators for inclusion

- The in operator
  - Does one string exist in another ?
  - For the in and not in case sensitivity matters
  - Example announcement = “The winners of the prize are Boris, Andy and Adam”
    - print(“Boris” in announcement) would produce true
    - print(“Charles” not in announcement) would produce true
  - in → Evaluates if object on left side is included in object on right side
  - not in → evaluates if object on left side is not included in object on right side
  - Can be used for strings and list, sets, tuples and dictionaries (for keys)
    - ‘this’ in ‘this is a string which produces true
    - 5 not in [1,2,3,4,6] which produces false

String Methods – Find and Index Method startswith and endswith count method

- find () : returns the lowest index where the substring or return -1 if not found
  - “browser”.find(“ow”) → 2
  - Accepts two arguments the substring to search for and starting index
  - “browserbrowser”.find(ow,3) → 9
- find() will tell that it exist and where. The in function will only tell you where.
- index() : if it cannot find the substring it will raise a ValueError
- startswith and endswith
  - startswith → Returns true if the string parameter is found at the start of the string
    - “Chuck was Here”.startswith(“Ch”)
  - endswith → Returns true if the string parameter is found at the end of the string
- count – The number time a substring is found in a string
  - “queuing”.count(“e”)) → 2

Capitialize, title, lower, upper and swapCase Methods

- story = “once upon a time”
- capitalize → return a new string with the first character capitalized
  - print(story.capitalize()) → Once upon a time
- title → return a new string where first letter of every word (space is delimiter) is capitalized
  - print(story.title()) → Once Upon A Time
- upper → All characters are capitalized
  - print(story.upper()) → “ONCE UPON A TIME”
- lower → returns a case where all lower case characters
- swapCase → returns a string where all the uppercase character are lowercase and lowercase character are uppercase
- Method chaining → Linking together several methods in sequence
- count → how many times the substring exist in the string
- find → Finds the index where the substring starts

split function

- Convert a string into a list
  - has parameters sep and maxsplit
    - maxsplit +1 is the number of arguments in the new list
  - Example
    - new_str = "The cow jumped over the moon."
    - new_str.split(' ', 3)
      - ['The', 'cow', 'jumped', 'over the moon.']
    - new_str.split('.')
      - ['The cow jumped over the moon', '']
    - new_str.split(None, 3)
      - **['The', 'cow', 'jumped', 'over the moon.']**
    - **new_str.split()**
      - **['The', 'cow', 'jumped', 'over', 'the', 'moon.']**

Boolean Methods For String

- islower() → returns true if the all the characters are lower
- isupper() → returns true if all the characters are upper
- isTitle() → true if the first character of each work uppercase and all the rest lowercase
- isAlpha() → true if all the characters are alphabetic
- isNumeric() → true if all the characters are numeric
- isalnum → true if the string has [a-zA-Z][0-9]
- Isspace() → true if string has all spaces
-

lstrip, rstrip and Strip Methods replace

- rstrip → strip the space on the right side
- lstrip → strip the space on the left side of the string
- strip → strip the space on the left and right side
- Each of the three functions has an extra parameter to specify what character you want to script
  - To remove all w from the beginning of the string : “www.python.org”.lstrip(“w”) → .python.org
  - To remove the w and the period of the string and the org → “www.python.org”.lstrip(“w.”) → python.org
  - To keep python only “www.python.org”.strip(w.org”) → python
- replace
  - “555 123 5555”.replace(“ “,”-”) → “555-123-5555”

Format Method

- Arguments by Relative Position
  - The object passed to the format method will be the order in which they are inserted
  - Example
    - mad_libs = {} laughed at the {} {}.”   // name, adjective, noun
    - print(mad_libs.format(“Bobby”, “green”, “alien”)) produces “Bobby laughed at the green alien”
  - If we fail to provide the correct number of index we get an IndexError
  - If we provide more arguments then it will run normally
- Pass the arguments by numeric position
  - Example
    - mad_libs = {0} laughed at the {1} {2}.”   // name, adjective, noun
    - print(mad_libs.format(“Bobby”, “green”, “alien”)) produces “Bobby laughed at the green alien”
  - Example
    - mad_libs = {2} laughed at the {1} {0}.”   // name, adjective, noun
    - print(mad_libs.format(“Bobby”, “green”, “alien”)) produces “Alien laughed at the green Bobby”
  -

Format Method

- Using argument with keyword parameters
  - Example
    - mad_libs = {name} laughed at the {adjective} {noun}.” // name, adjective, noun
    - print(mad_libs.format(name=“Bobby”, adjective=“green”, noun=“alien”)) produces “Bobby laughed at the green alien”
  - Example
    - mad_libs = {name} laughed at the {adjective} {noun}.” // name, adjective, noun
    - print(mad_libs.format(name=“Bobby”, adjective“green”, noun=“alien”)) produces “Alien laughed at the green Bobby”
  - The intent of the string is better understood.
- Example
  - name = input(“Enter a name: “)
  - adjective = input(“Enter an adjective: “)
  - noun = input(“Enter a noun:) “
  - mad_libs = {name} laughed at the {adjective} {noun}.” // name, adjective, noun
  - print(mad_libs.format.format(name = name ,adjective = adjective, noun = noun))

Formatted String Literals (f-strings)

- Example
  - name = input(“Enter a name: “)
  - adjective = input(“Enter an adjective: “)
  - noun = input(“Enter a noun: “
  - mad_libs = f”{name} laughed at the {adjective} {noun}.” // name, adjective, noun
    - The “f” can be lower or upper case
  - print(“mad_libs”) → Bobby laughed at the Green Alien
  - print(f” 2 + 2 = { 2 + 2 }” → “2 + 2 = 4” // expression directly in the string be printed out

The If-Statement
The bool Function (Truthiness and Falseness)

- Example
  - If 5 > 3:
    - print(“Will be true”)
- When an if is true the following block is executed
- The bool Function (Truthiness and Falseness)
  - Truthiness:
    - Any Number other than 0
    - Any Non Empty String
    - Anything not in the list below.
  - Falseness:
    - 0 , 0.0m 0j , Decimal(0), Fraction(0,1)
    - Empty String, (), [], {}, set(), range(0)
    - None
    - False
- Bool() : Convert the input into an equivalent string
  - Example print(bool(1), bool(0)) will produce True, False

Else statement
Conditional Expressions

- Example
  - if 20 > 15:
    - print(“This is true”)
  - else
    - print(“This is false)
- The elif Statement
  - if (20 > 15):
    - print(“This is true”)
  - elif (20 > 0):
    - print(“This is true, but the elsif caught it”)
  - else:
    - print(“This is false”);
- Conditional Expressions
  - zip_code = “20121”
  - check = “Valid” if len(zip_code) == 5 else “Invalid”

Recursion

- Each recursive equation has a base case and a call to itself
- A function that calls itself
- String reversal
  - A string of the length of 1 is the same backwared as forwards (this is the base case)
  - Get the last character of the string
  - Example
    - Straw W + reverse(stra) W + a + reverse(str) W + a + r + reverse(st) W + a + r + t + reverse(s)
    - W + a + r + t + s
- Example
  - Def reverse(str)
    - If len(str) < 1:
      - Return str
    - return str[-1] + reverse(str:-1])

And, Or, Not Keyword
While Loop

- Example
  - If 5 < 7 and “rain” == “rain”:
    - print(True)
- and, or, not are all short circuited.
- Example
  - If 90 < value < 100:
    - print(“The value is in range”)
- While loop
  - count = 0
  - while count <= 5:
    - print(count)
    - count +-=1
- break Terminates a for or while loop
- continue Terminates one iteration of a for or while loop

For Loop

- Example for loop
  - cities = [“new york city”, “mountian view”, “chicago”, “los angeles”]
  - for city in cities:
    - print(city.title())

Modules: Scripts Modules and the import keyword

- A module is any python file with .py extension considered by the community as scripts
- The python community describes a module as python file that is meant to be use by other files
- A script is a python files that is meant to be executed directly
- Example
  - calculator.py
    - creator = “Boris”
    - pI = 3.14
    - def add(a,b):
      - return a + b
    - def sub(a,b):
      - return a – b
    - def area(radius):
      - return PI * radius * radius
      - print(add(1,3)
  - Each module creates a namespace around its names
  - A module is an object that represent a collection of names under a shared name space.
  - Python will only import the file once.

Modules: Scripts Modules and the import keyword

- Example
  - my_program.py
    - Import calculator
    - print(calculator.creator)
    - print(calculator.PI)
  - When python import a module it will execute all the code in that module.
  - Example
    - other_scirpt.py
      - num = (2 +3)
    - demo.py
      - import other_script
      - x = 5 + other_script.num // access the variable from the other script
      - print(4)
  - This part package are always placed

Modules: The Python Standard Library (The String, math and Module this)

- Example
  - import string
  - print(string.ascii_letters)
  - print(string.ascii_lowercase)
  - print(string.digits)
  - print(string.whitespace)
  - print(string.capwords(“hello there”) would produce “Hello There”
- Example
  - import this
  - When executed the python manifesto will appear
- When the import cannot be found an ImportError is thrown
- If the name after import has the .py as part of the name a ModuleNotFoundError is thrown
- If the module is passed into the type function the class is module import string; print(type(string)) → String
- dir contains the list of names found in the module namespace

Modules: The __name__ special variable

- Example
  - Import math, calculator   // Not recommended, but can be done
  - Print(math.__name__) would produce math
  - print(calculator.__name__) would produce calculator
  - print(__name__) would produce __main__
- how __name__ works
  - When running a script python set the __name__variable
  - If the file is the launching point of the program then the name will __main__
  - In the module contains code outside a function print(__name__) then the name will be the name of the module
  - If the file is execute as module, it will provide the name of the file
    - __name__ tells us if its being executed as a script or module.
- If __name__ == “__main__”
  - good to put statements that you don’t want exported for example test code

Modules: Alias with the As Keyword

- Example
  - import calculator as calc
  - Import datetime as dt
  - print(calc.add(3,5))
- Import Specific Attributes with the form Syntax
  - Want to import the modules names directly into the file’s namespace
  - Example
    - from calculator import creator, PI   // Get the variables from the module
    - from math import sqr
    - print(creator)
  - Increases the chances of names collisions
- I

Modules: Import Specific Attributes with the from Syntax

- import all attributes with * syntax
  - From calculator import * // import all public attributes to the current namespace, could overwrite or be overwritten by objects with the same name
  - print(add(3,5))   // We are not adding the module namespace, but increasing collisions
- In the _calculator module insert _year (The underscore tells python not to export the variable))
  - If the module is imported and _year used an import error will be raised since it is not defined
- Other examples
  - To import an individual function or class from a module
    - from module_name import Object Name
    - example – from collections import defaultdict
      - In the code only refer to it as defaultdict
  - To import multiple individual objects from a module
    - from module_name import first_object, second object
  - To Rename a module
    - import module_name as new_name
    - good when two modules have the same name
  - To import an object from a module and rename it
    - from module_name import object_name as new_name
  - To use all object from a module, use the standard import module_name and access with dot notation
    - import module_name

Modules : Packages and Name

- In order to manage code better modules in the python library are split into sub-modules
  - A package is module that contains sub-modules
  - A sub-module is specified with the usual dot notation
  - Example : import package_name.submodule_name
  - example os.path // where path is the submodule
  - example
    - import os
    - path.isdir(‘/’)
    - import os.path
    - isdir(‘/’)
    - from datetime import datetime
-

Modules: __init__.py

- A directory is declared add a file __init__.py file
- Not mean to be script files, but python runs them automatically when the package is loaded
- Any *.py found that directory with the __init__.py the __init__ will be executed
- W gave the directory structure
  - Project
    - feature
      - __init__.py
      - copyright.py
- With the __init__.py it will be run and it will get executed.
  - Import feature.copyright // copyright.py in the same directory
  - print(feature.copyright.date_of_copyright)
- If you have directories inside the directory with the __init__py the parent directories __init__.py will get executed
- From feature.subfeature.calculator import subtract

Modules: __init__.py 2

- Common the use case for imports that are nested.
- Problem to import feature.subfeature.calculator
  - print(fature.subfeature.calculator.subtract(10,-1))
- Fix
  - __init__.py
  - from .calculator import creator, PI, add, subtract, area
    - .calculator will be in the file with the calculator.py
    - The name will be export outside and the names will get them from the subfeature package.
      - Advantage a lot easier to remember
    - Now we can
      - Import feature.subfeature
      - print(feature.subfeature.subtract(10,-1))
  -

File: Reading a file with the open function and Read Method
Read Line by Line

- Example
  - cupcakes_file = open(”cupcakes.txt”, “r”)   // accepts two arguments filename and mode (do we read, write append)
  - close(cupcakes)
  - Not the best approach the close method may not run if there is an error
    - solution with context → A wrapper that covers the operation. If something does not work the wrapper will close the file not matter what
- Example
  - with open(”cupcakes.txt”, “r”) as cupcakes_file: // with performs automatic clean up when the block is done executing
    - content = cupcakes_file.read()   // reads all the data in the file as a string
- If filename does not an exist get FileNotFoundError
- Read Line by Line
  - Example
    - with open(“cupcakes.txt”) as file_object: // Read each line
      - for line in file_object:
        - print(line.strip())   // The strip removes the new line charactrer
  - The print function add a line break. To get rid of use the rstrip method
- Also close the files because too many open files will cause the system to run out of file handles which throws OSError
- readline() – Read each line (end with \n) and if no more lines then returns an empty string

File: Write to a file
Append to a file

- Example
  - file_name = “my_first_file”
  - with open(file_name, “w”) as file_object:
    - file_object.write(“Hello File!\n”)
    - file_object.write(“Second line”)
- Append a file
  - with open(file_name, “a”) as file_object:
    - file_object.write(“The third line has been appended”)
- If the file does not exist for the write or the append Python and run the program
  - With write it will create the file every time getting rid of the data. All previous data in the file will be deleted.
  - With append it will create the file the first time and keep adding the data to it.
- Example of a read
  - f = open(‘/my_path/my_file.txt’, ‘r’)
  - file_data = f.read() // Without no arguments reads the whole file and with an integer argument read that number of characters
  - f.close()

Exception

- Common Exception
  - ValueError
    - An object of the correct type, but inappropriate value is passed as input ot a built in operation or function
  - AssertionError
    - An assert statement fails
  - Index Error
    - A sequence subscript is out of range
  - Key Error
    - A key can’t be found in a dictionary
  - TypeError
    - An object of an unsupported type is pass as input to an operation or function
  - Unbound local error
    - Trying to access a local variable before it is defined. Make sure local scope of variable in function is defined or value assigned to it.
  - NameError
    - Identifier is not found in the local or global namespace. Make sure the reference to the identifier. Make sure the reference to the identifier is correctly added to the code
  - Assignation error
    - Inconsistency in how many values being unpacked and how many variables should be assigned to.

Exception Handling:Introduction to Error Exceptions
Try Except Block

- An exception is a special object that Python uses to manage error during program execution
- A traceback is a report of the exception that was raised
- Example – Try Except Block
  - def divide_five_by_number(n):
    - try:
      - return 5/n
    - except:
      - pass   // When n is 0 then None will be returned since 5/n throws the exception
  - print(divide_five_by_number(0))   // Causes a ZeroDivsionError
- Example – Try Except Block
  - def divide_five_by_number(n):
  - try:
    - calculation 5/n
  - except: /   // Respond to any error later on we have it respond to a specific error in a later chapter
    - calculation = 5   // Fixes the value
  - print(divide_five_by_number(0))   // Causes a ZeroDivsionError

Exception Handling:Catching One or more Specific Exceptions

- Example
  - def divide_five_by_number(n):
    - try:
      - return 5/n
    - except ZeroDivisionError: // Enter the except block if a number is divided by 0
      - return “You can’t divide by zero!”
    - except TypeError as e:
      - return f”No dividing by invalid objects! {e}”
    - return calculation
    - print(divide_five_by_number(0))   // Causes a ZeroDivsionError
- Example
  - def divide_five_by_number(n):
    - try:
      - return 5/n
    - except (ZeroDivisionError, TypeError) as e: // catching multiple exceptions with the same except
      - return f”No dividing by invalid objects! {e}”
    - return calculation
- print(divide_five_by_number(0))   // Causes a ZeroDivsionError

Exception Handling:The raise keyword

- Example
  - def add_positive_numbers(a,b):
    - try:
      - if (a <=0 or b <= 0):
        - raise ValueError(“Both numbers must be positive”)   // Throw an exception. The message in quotes is optional
      - return a + b
    - except ValueError as e:
      - return f(“Caught the ValueError: {e}”)

Exception Handling:User Defined Exceptions

- All native exception are found in a hierarchy

    The Base Exception is BaseException

    - which has two children
        - Exception which has many children
        - KeyboardInterrupt which has no children
- Example
    - def class NegativeNumberError(Exception):
        - “””One or more inputs are negative”””
        - pass
    - def add_positive_numbers(a,b):
        - try:
            - if (a <=0 or b <= 0):
                - raise ValueError(“Both numbers must be positive”)
                - return a + b
        - except NegativeNumberError
                - return “Shame on you, not valid”

Exception Handling:Exception Inheritance Hierarchies

- Pattern : For a module that can create many exceptions
  - create a base exception class for that exceptions defined for that module
  - Subclass that class to create specific exception
- Example
  - 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 Stupd Mistake”)
  - except Mistake as e: // Will catch Stupid Mistake or Silly Mistake
    - print(f”Caught the error: {e}”)

Exception Handling: The Else and Finally Block

- An else block will execute if the try block executes without error
- The finally block will run no matter what
- Example
  - x = 10
  - try:
    - print(x+5)
  - except NameError: // Can have multiple blocks for different exceptions
    - print(“Some variable is not defined!”)
  - else: // If the program runs into no exceptions
    - print(“This will print if there is no error in the try”)
  - finally: // Runs whether an exceptino is thrown or not
    - print(“This will print whether an exception has been thrown or not.”)
  - All exception not handled by the developer using except will be displayed on the console
  - Address more than one type of exception
    - except (ValueError, KeyboardInterrupt): (ValueError, KeyboardInterrrupt) is a tuple
      - # some code