Python – Collections

Learn to Code with Python

Intro to the List

- A list is an object that stores an order sequence of object
  - Can have many different type of variables
  - Can be modified where a string cannot
- empty = list() // Another way to create an empty list , but used []
- empty = [“Coke”, “Root Beer”, “Sprite”]
  - each element in a list can be a different type
- len() → used to get the size of the list
  - empty.len() = 3
- Lists are mutable data structures.
- Select a List Element by Positive or Negative Index
  - (“lunch” in [“breakfast”, “lunch”, “dinner”] } produces true
  - (“lunch” not in [“breakfast”, “lunch”, “dinner”] } produces true
  - Example [1,2] in [1,2,3] means that is it searching for element [1,2]
- IndexError : list index out of range – The index was greater than the maximum index of the list.

Select a List Element by Positive or Negative Index Slice Multiple Elements from a List

- Example
  - [“Chrome”, “Firefox”, “Safari”, “Opera”][0] will produce Chrome
  - [“Chrome”, “Firefox”, “Safari”, “Opera”][10] will raise an IndexErrorException
  - [“Chrome”, “Firefox”, “Safari”, “Opera”][2][1] will produce an “a” from safari
  - [“Chrome”, “Firefox”, “Safari”, “Opera”][-1] will produce Opera
- 0 based index
- Slice Multiple Elements from a List
  - Example
    - [“Biceps”, “Triceps”, “Deltoid”, “Sartorius”][1:3] produces [“Biceps”, “Triceps”]
    - [“Biceps”, “Triceps”, “Deltoid”, “Sartorius”][::-1] produces [“Sartoius”, “Deltoid”, “Triceps”, “Biceps”] // reverse a list]
  - Works the same with Slicing for Strings

Introduction to the for-loop Interaction with Conditional Logic

- Example
  - dinner = “Steak and Potatoes”
  - Each for loop has a block
  - For character in dinner:
    - print(character)
  - Character can be used outside the scope of the for loop and will have “s” as its value
-

Iterate in reverse with the reversed function

- Example
  - the_simpsons = [“Homer”, “Marga”, “Bart”]
  - For person in the simpsons[::-1]: // Reverses a list
    - print(character)
- There is a builtin reverse function which returns a generator object that can be iterated over. You do not get a list object returned
  - Generator a better for huge list since it process the list one line at a time instead all at once
- Example
  - print(reverse(“the_simpsons”)) The type you get back from reverse is a list_reverseiterator for list

Enumerate Range

- Example
  - Returns the index and value at the index into different varibles
  - errands = [“gym”, “lunch”, “promoted”, “sleep”]
  - print(enumerate(errands)) enumerate returns an enumerated object
  - For index,errand in enumerate(errands):
    - print f”{index} = { errand }” // The first index printed will be 0
      -
  - For index,errand in enumerate(errands, 1):
  - print f”{index} = { errand }” // The first index printed 1 because of the second parameter
- range
  - The range function is a generator object that returns a Range Object
  - for number in range(5):
    - print(number) // will produce 0,1,2,3,4
  - range(3,9) // would produce 3,4,5,6,7,8
  - range(10, 101, 10) // would produce 10,20,30,40,50,60,70,80,90,100
  - Range(99, 1, 10) // would produce 99,88,77,66,55,44,33,22,11,0
  - print(list(range(0,-5)) // would produce []

Break, Continue
Command Line Arguments

- The break terminates a for loop before it completed the iteration of the list
- The continue keyword terminates the current loop
- Command Line Arguments
  - Example
    - import sys
    - print(sys.argv) // argv : A list of all arguments from the command line
    - print(type(sys.argv)) // The type is a list
  - The first argument of argv is the filename
-

Assign a new value at Index Position
Assign new values to a list slice
Append Method

- Example
  - crayons = [“black”, “white”, “green”] → crayons[1] = “green” and crayons[-1] = “green”;
- Cannot use the [] to assign something to the end of the list. In reality we would get an IndexError
- Assign new values to a list slice
  - coworkers = [“Michael”, “Jim”, “Dwight”, “Pam”, “Creed”, “Angela”]
  - [“Michael”, “Jim”, “Dwight”, “Pam”, “Creed”, “Angela”][3:5] = [“Chuck”, “Sharon”] // Replace Pam and Creed with Oscar, Ryan
  - [“Michael”, “Jim”, “Dwight”, “Pam”, “Creed”, “Angela”][3:5] = [“Chuck”] // Replace Pam and Creed with Oscar
  - [“Michael”, “Jim”, “Dwight”, “Pam”, “Creed”, “Angela”][3:4] = [“Chuck”, “Sharon”] // Replace Pam with Chuck and Sharon
- append
  - Counties = [“US”]
  - Countries.append(“Mexico”)
  - Example spices = ["paprika", "nutmeg", "ginger", "cinnamon", "turmeric"] followed by spices.append(["garlic", "berbere", "sansho"])
    - ["paprika", "nutmeg", "ginger", "cinnamon", "turmeric", ["garlic", "barbere", "sansho"]]
  - names[“Grarcia” “O’Kelly”, “Davis”]
    - print(“-”,join(names)) would produce GarciaO’KellyDave // Don’t forget to add commas to the list

Extend Method

- Accepts a list of elements and add them to the list that the method is invoke on
- Example – No new list is created numbers has all 5 values at the nd
  - numbers = [1,2]
  - numbers.extend[3,4,5]
  - print(numbers) prints [1,2,3,4,5]
- Example + sign can combine two list, but creates a new list
  - numbers1 = [1,2]
  - numbers2 = [3,4,5]
  - numbers = numbers1 + numbers2
  - print numbers would produce [1,2,3,4,5]

Insert Method
Pop Method

- Insert an element at a given index method
- Example
  - numbers = [1,2,3,4,5]
  - numbers.inert(2, 2.5)
  - print(numbers) would produce [1,2,2.5,3,4,5]
- Example
  - numbers.insert(10,8) would produce [1,2,2.5,3,4,5,8]
- Pop Method
  - Example
    - numbers = [1,2,3,4,5]
    - number = numbers.pop() // Number would be 5
    - print(numbers) // prints [1,2,3,4]
    - numbers.pop[2] // would produce [1,2,4]
    - numbers.pop[-2] // would prodcue [1,4]

Del Keyword clear method reverse

- The Del keyword does not return the element (cannot save the element) and allows the use of slicing
- Example
  - numbers = [1,2,3,4,5]
  - del numbers[3] // produces [1,2,3,5]
  - del source[-1] // produces [1,2,3]
  - del sources[0:2] // produces [3]
- removes all elements from the array
- clear → Remove all elements in the place
- reverse method→ reverses the order of the list in place

Sort()

- sorts the elements and returns a copy of the list using the original list alone
- Example
  - numbers = [3,1,2,4,5]
  - numbers.sort()
  - numbers.reverse()
  - sorted(numbers) // Returns a new list with the values sorted
- Capital letters are sorted before lower case letters

Count Method
Index Method

- The number of times the element appear in the list
  - Example
    - numbers = [1,1,2,3,4,1,5]
    - Numbers.count(1) would produce 3
- The index method
  - Example
    - numbers = [1,1,2,3,4,1,5]
    - numbers.index(2) = 2 // 2 is at index of 2
    - numbers.index(1) // 0 is at first index of 1. It will return the first index if it is found multiple time in the list
    - numbers.index(8) // Returns an index value error. Can use the in operator to verify the value is in the list
    - numbers.index(1, 3) // 5 is at first index with 1 after the start index of 3
    - numbers.index(1,1) // 1 is at first index with 1. The search start with 0 or the second parameter as the number.

Copy Method
Split Method of String

- Does a shallow copy of the list
- Example
  - numbers = [1,2,3,4,5]
  - numbers_copy = numbers.copy()
  - print(numbers_copy) // The value printed is [1,2,3,4,5]
  - even_more_numbers = numbers[:] // Same as copy command
- Split Method of String
  - Example
    - users = “Bob, Dave, John”
    - print(users.split(“, “) // The output will be a list with three values
    - print(users.split(“, “, 1) // The Second parameter is the max number of time the string will be split. The output is ‘Bob, ‘Dave John’
  - Raises a value error If the string is empty

Join
Zip

- Example
  - address = [“500 Fifth Avenue”, ”New York”, “NY”, “10036”]
  - print(“, ”.join(address)) // The String is 500 Fifth Avenue, New York, NW, 10036
  - can use a \n and get each element on the a different line
  - error if you try ot join anything, but string.
  - example : names[“Carol”, “Albert”, “Ben”, “Donna”] then print(“ & “.join(sorted(names)) prroduces Albert & Ben & Carol & Donna
- Zip
  - Returns Iterable Objects
  - Combine elements across multiple list based on common index positions returns a zip object
  - Actually returns an ignorable generator object that from each list at the same time based on a shared index position
  - Example
    - list1 =[“1”, “11”, “21”]
    - list2 = [“2”, 12”, “22”]
    - List3 = [“3”, “13”, “33”]
    - zip(list1, list2, list3)
    - print(list(zip(list1,list2,list3)) // Returns [(“1”, “2”, “3”), (“11”,”12”,”13”), (“21”,”22”,”23”)]
    - For (number1, number2, number3 in zip(list1,list2,list3)
      - Print (f”{number1}, {number2}, {number3}”)

Cool zip Example

- Using zip to zip a group of list
  - from string import Template
  - x_coord = [23, 53, 2, -12, 95, 103, 14, -5]
  - y_coord = [677, 233, 405, 433, 905, 376, 432, 445]
  - z_coord = [4, 16, -6, -42, 3, -6, 23, -1]
  - labels = ["F", "J", "A", "Q", "Y", "B", "W", "X"]
  -
  - iterations = len(x_coord)
  -
  - points = list(zip(labels, x_coord, y_coord, z_coord))
  -
  - answer_template = Template("$label: $x, $y, $z")
  - for point in points:
  - print(answer_template.substitute(label=point[0], x=point[1], y=point[2], z=point[3]))
-

Cool Zip Example

- Using zip to add each create list to an array
  - x_coord = [23, 53, 2, -12, 95, 103, 14, -5]
  - y_coord = [677, 233, 405, 433, 905, 376, 432, 445]
  - z_coord = [4, 16, -6, -42, 3, -6, 23, -1]
  - labels = ["F", "J", "A", "Q", "Y", "B", "W", "X"]
  -
  - points = []
  - for point in zip(labels, x_coord, y_coord, z_coord):
  - points.append("{}: {}, {}, {}".format(*point))
  -
  - for point in points:
  - print(point)

cool zip examples

- Zip list to a dictionary
  - cast_names = ["Barney", "Robin", "Ted", "Lily", "Marshall"]
  - cast_heights = [72, 68, 72, 66, 76]
  - cast = dict(zip(cast_names, cast_heights))
  - print(cast)
- Unzip tuples
  - cast = (("Barney", 72), ("Robin", 68), ("Ted", 72), ("Lily", 66), ("Marshall", 76))
  - names, heights = zip(*cast)
  - print(names)
  - print(heights)
- Transpose withy zip (convert a 4 by 3 matrix into a 3 by 4 matrix)
  - cast = ["Barney Stinson", "Robin Scherbatsky", "Ted Mosby", "Lily Aldrin", "Marshall Eriksen"]
  - heights = [72, 68, 72, 66, 76]
  - for i, character in enumerate(cast):
  - cast[i] = character + " " + str(heights[i])
  - print(cast)

Multidimensional Lists

- Example
  - bubble_tea_flavors[
    - [“HoneyDew”, “Mango”, “Passion Fruit”],
    - [“Peach”, “Plum”, “Strawberry”, “Taro”]
    - [“Kiwi”, “Chocolate”]
  -]

List Comprehension (Part 1 and Part 2)

- A new list is always created
- list comprehension can create a list from any iterable object
- Basic Setup can have an expression, for loop and option if statement
- Example
  - numbers = [3, 4, 5, 6, 7]
  - squares = [number ** 2 for number in numbers]
- Example
  - rivers = [“Amazon”, “Nile”, “Yangtze”]
  - len_of_strings = [len(river) for river in rivers]
- Example
  - [“abcdefghijklmnopqrsstuvwxyz”.index(char) for char in “donut”]
  - even [number / 2 for number in range(20)]
- Example
  - donuts = [“Boston Creme Donut”, “Jelly”, “Vanilla Create”]
  - creamy_donut_string_length = [len(donut) donut for donuts in donuts if “Cream” in donut]
  - creamy_doughnut_first_word = [donut.split(“ “) for donut in donuts if “Cream” in donut])

Filter Function
Map Function

- Extract a subset values based on a condition being met.
- Example
  - animals = [“elephant”, “horse”, “cat”, “giraffe”, “cheetah”, “dog”]
  - def is_long_animal(animal):
    - return len(animal) > 5
  - print(filter(is_long_animal, animals)) returns a filter object
- map function – Invokes the function every element in the list
  - functions are first class objects in Python can pass them in and/or return them.
  - example
    - numbers = [4,8,15,16,23,42]
    - def cube(number):
      - return number ** 3;
      - print(map(cube, numbers))
      - numbers = [“cat”, “bear”, “zebra”, “donkey”, cheetah”] // returns a map object
      - print(map(len, animals));

Lambda Function
All and Any Functions

- An anonymous function without a name. Usually used in only one place
- Example
  - metals = [“gold”, “silver”, “platinum”, “palladium”]
  - print(filter(lambda metal: len(metal) > 5, metals) // Implicit Return
  - print(list(map(lambda word: “p” in word , metals))) // find all meta with the word p
  - lambda x,y: x * y
- All and Any Functions
  - all → A function that accepts a list of values. If all are truthy then returns true
    - print(all [1,2,3]) True
    - print(all[0,1,2,3]) False (0 is considered falsely)
    - print(all []) True
  - any → A function that accepts a list of values. If one or more of the values is true then returns true
    - print([“ “, “”) True since “ “
    - print(any([“”])) false

Min and Max
Sum
Dir

- Max or Min accept a iterable or sequential arguments
  - max element in a list of number is the max integer , max string is the string that would occur last if sorted alphabetically
  - The max function is defined in terms fo the > operator
  - Example
    - print(max(3,5,7,9)) returns 9
    - print(max(“D”, “Z”, “K”)) returns the string closes to the end of the alphabet
    - print(min(“D”, “Z”, “K”)) returns the string closes to the beginning of the alphabet
    - print(max(“A”,”a”,”Z”,”z”)) returns “z”
    - print(min(“A”,”a”,”Z”,”z”)) returns A
- Sum
  - A list of values and gets the sum back
  - print(sum([2,3,4]) → 9
- Example
  - print(dir([])) produces all the funcitnos for the list
  - dir(“pasta”) All methods for a String Object.
- __hash__ A standard means a function should be considered private
- pasta.__dir__() invokes the private dir function

Format function

- Presentation of a numeric value
- Example
  - number = 0.123456789
  - print(format(number, “f”)) // Produces 0.1234567 and the type is a string
  - print(format(number, “.2f”)) // Produces 0.12
  - print(format(0.5, “%”)) // Produces 50.000000%
  - print(format(0.5, “.2%”)) // Produces 50.00%
  - print(123456, “,”) // 123,456
  -

Tuples

- AN ordered fixed length immutable list
- tuples can hold heterogeneous data types.
- Example – Both are tuples
  - foods = (“Sushi”, “Steak”, “Guacamole”).
  - foods = “Sushi”, “Steak”, “Guacamole” The comma operator is what makes it a tuple
- For an empty tuple parenthesis are required
  - empty = ()
- Example
  - mystery = (1)
  - print(type(mystery)) type will be an int not a tuple
  - mystery = (1,)
  - print(type(mystery)) type will be a tuple not an int.
- Example
  - tubple_a = 1,2 and tuple_b = (1,2); print(tuple_a == tup;le_b) and print(tuple_a[1] // True, 2

List vs Tuples

- len(tuple) will produce the number of element in the tuple
- print(birthday[0]) will produce the first element
- If a birthday tuple has 6 elements and you write birthday[7] will produce indexError
- print(birthday[-1]) will produce the last element
- birthday[1] = 13 will produce TypeError
- The list inside the tuple are mutable
  - address = (
    - [“Hudson Street’, ‘New York’, ‘NY’].` // The data inside the list can be modified
    - [“Franklin Street’, ‘ San Francisco’ , ‘CA’]
  - }
-

Unpacking a Tuple

- Example
  - employee = “Charles”, “Stockman”, “Senior Software Engineer”, 53
  - first_name, last_name, position, age = employee
- Example – Unpacking a list
  - subject, verb, adjective = [“Python”, “is”, “fun”]
  - length, width, height = 52, 40, 100
- Errors
  - first_name, last_name, title = employee // Get a ValueError – To many errors to unpack
  - a,b,c,d,e = employee // Get a Value Error – To many errors to unpack
- Example – Swap Variables
  - a = 5
  - b = 10
  - b, a = a , b
-

Unpacking a Tuple 2: Using * to Destructure Multiple Elements

- Example
  - qualities = ("Determination", "Grit", "Perseverance", "Optimism", "Excitement")
  - traits, *skills, characteristics = qualities
  - print(type(skills)) **// Will produce <class 'list'>**
- Example
  - employee = “Charles”, “Stockman”, “Senior Software Engineer”, 53
  - first_name, last_name, *details = employee would produce “Charles”, “Stockman”, [“Senior Software Engineer”, 53]
  - *names, position, age = employee would produce [“Charles”, “Stockman”], “Senior Software Engineer”, 53
  - *names, position, *age = employee Not permitted
  - name, *details, age = employee ”Charles”, [“Stockman”, “Senior Software Engineer”] , 53
- Can use the asteix syntax only once for each line on the left hand side
-

Variable Number of Function Arguments with *args

- We provide an asterix symbol before a parameter into a tuple of values.
  - The convention is args, but the name is your decision
- Example
  - def accept_args(*args):
    - print(type(args)) **// Class is tuple**
    - print(args) // All values passed in as a tuple.
  - accpet_args(1,3,5)
  - accept() // Args would print ()
- Add additional parameters best to add them to the left
  - def accept_args(count, *args)
  - accept_args(1,3,9,7,8,-14) // count = 1 args = (3,9,7,8,-14)
- Add additional parameters after the *args
  - def accept_args(*args, nonsense = “Shazam”):
  - print accept_args(1,2,3,4,5,6, “Hoorah”)

Variables, Objects and Garbage Collection

- A variable
  - used to identify the data
  - does not have a data type
- Example
  - a = 10
    - a is a name
    - 10 is the object
      - it has a datatype
- Garbage collection
  - The cleaning object that no longer have a name (reference)
  - Example
    - a = [1,2,3]
    - a= [4,5,6]
    - The result will be garbage collected [1,2,3]

Shared References with Immutable and Mutable Types

- Example
  - a = 3
  - b = a
  - Both a and b are referencing the same variable. The variables a and b are not linked to each other.
- Immutable Objects: Numbers, Strings, Booleans, Tuples.
- mutable type can have its state modified
  - Example
    - a = [1,2,3[
    - b= a // A, B are a reference to the same list
    - a.append(4)
    - print(b) // Would produce [1,2,3,4]

Equality vs Identity

- Equality : Are two Objects equal (same values object and shape)
- Identity : Are the two names point to the same object
  - use the is keyword to evaluates if both side have the same identity
  - is not is the keyword evaluates if both sides have different identitites
- Example
  - students = [“Bob”, “Sally”, “Sue”]
  - athletes = students
  - nerds = [“Bob”, “Sally”, “Sue”]
  - Equality
    - print(students == athletes) // True since students is the same list
    - print(students == nerds) // True since nerds is the same list
  - Identify
    - print(students is athletes) // Is the list the same : yes (referencing the same object)
    - print(students is nerds) // is the list the same : no (These are two different memory locations)
  -

Equality vs Identity

- immutable types
  - Integer, Float, String, Boolean → does not create multiple object in memory
  - Example
    - a = “hello” // “hello” is only stored in memory once
    - b = “hello”
    - c = “hello”
  - Example
    - a = 1 a and b reference the same object
    - b = 1
    - print (a == 1) // Would produce true
    - print (a is 1) // Would produce true
- Example
  - a = [1 ,2 ,3] b = a c = [1,2,3]
  - a == b (true), a is b (true) a == c (true) a is c (false)
  -

Shallow and Deep Copies

- Shallow Copy
  - Create a shallow copy
    - list slicing
    - copy method
    - copy function from the copy module
  - Example
    - a = [1,2,3]
    - b = [:]
    - print (a == b) True Same Shape, Same Data Type
    - print (a is b) False Not the same memory location
  - Example
    - b = a.copy()
    - print (a == b) True Same Shape, Same Data Type
    - print (a is b) False Not the same memory location
  - Example
    - import copy
    - b = a.copy()
    - print (a == b) True Same Shape, Same Data Type
    - print (a is b) False Not the same memory location

Shallow Deep Copies

- Example – copy Top Level Elements in the list
  - numbers = [2, 3, 4]
  - a = [1, numbers, 4]
  - Could do a shallow copy (ex. b = a[:])
  - print(a[1] is b[1]) will be true
  - a[1].append(100)
  - print b will produce [1, [2,3,4.100] ,4]
- Deep Copy – Copy all elements in the list
  - numbers = [2, 3, 4]
  - a = [1, numbers, 4]
  - b = copy.deepcopy(a)
  - print(a[1] is b[1]) will return false
  - a.append(100)
  - print(b) will produce [1, [2,3,4], 4] // The append to list a does not have an effect on list b

Intro to Dictionaries

- Dictionary – A mutable data structure with key/values where keys are unique and values can be duplicates
- keys are an immutable datatypes like integers, or tuples or strings, float, bool, tuplel, but not lists or sets
- A value can be String, List, Set, Dictionary, map
- A dictionary is used fro mappings and list are used for order
- Example
  - ice_cream_preferences = {
    - “Benjamin”: “Chocolate”,
    - “Sandy”, “Vanilla”,
    - “Marv”: “Cookies & Cream”,
    - “Julie” : “Chocolate”
  - }
  - ice_cream_preferences[“chuck”] would produce a key error
- The len(ice_cream_preferences) = 4
- print(“Benjamin” in ice_cream_preferences) would print true
- pirnt(“Chuck” not in ice_create_preferences) would print true
- Can check for none by using <var> is None or <var> is not none

Access a Dictionary Value by Key or the get Method

- Example
  - flight_prices = {
    - “Chicago”: 199,
    - “San Francisco”: 499,
    - “Denver”: 295
  - }
- print(flight_prices[“Chicago”]) // Would produce 199
- If a key does not exist in the dictionary then a key error will be raised.
- keys can be any immutable data type
- Example
  - gym_membership_packages = {
    - 29: [“Machines”],
    - 49:[“Machines”, “Vitamin”]
    - 79::[“Machines”, “Vitamin”, “Suana”]
  - }

Access a Dictionary Value by Key or the get Method

- get :
  - A method of the dictionary class
  - Two parameters : the key and the value to return if the key is not present.
  - Returns either the or value to return if the key is not present
  - Guarantee a KeyError is not thrown
- Example
  - Use the previous gym_membership_packages
  - print(gym_membership_packages.get(29, [“Basic Dumbbells”]) returns [“Machines”]
  - print(gym_membership_packages.get(100, [“Basic Dumbbells”]) returns [“Basic Dumbbells”]
  - print(gym_membership_packages.get(100) returns None
- example – Compound Map (accessing an element and adding an element)
  - elements = { “hydrogen” : { “number” : 1, “weight” 1.00794, “symbol”, “H” }
    - elements[“hydrogen”][“symbol”] = “H”
  - oxygen = { “number”:8, “weight” : 15.999, “symbol” : “O” } ; elements[“oxygen”] = oxygen

The in and not in Operators in a Dictionary

- Example
  - pokemon = {
    - “Fire”: [“Charmander”, “Charmeleon”, “Charizard”],
    - “Water”: [“Squirtle”, “Warturtle”, “Blatoise”],
    - “Grass”: [“Bulbasaur”, “Venusaur”, “Ivysaur”]
  - }
- print(“Fire” in pokemon) // Returns True
- print(“Electric” in pokemon) // Returns False
  - if “Zombie” in pokemon: // avoids exception being thrown
    - print(“Zombie”)
  - else
    - print(“not present”)

Add or Modify Key-Value Pair in Dictionary

- Example
  - sports_team_rosters = {
    - “New England Patriots”: [“Tom Brady”, “Rob Gronkowski”, “Julian Edelman”],
    - “New York Giants”: [“Eli Manning”, “Odell Beckham”]
  - }
  - sports_team_rosters[“Pittsburgh Steelers”] = [“Ben Roethlisberger”, Antonio Brown”] // Add the key/value to the dictionary
  - sports_team_rosters[“New York Giants”] = [“Eli Manning”] // Removes Odell from dictionary
  -
  - video_game_options = dict() // Creates a new dictionary
-

Add or Modify Key-Value Pair in Dictionary

- The key can not be a dynamic valiues
  - words =[“danger”, “beware”, “danger”]
  - def count_words(words):
    - counts = {}
    - for word in words:
      - if word in counts:
        - counts[word] += 1
      - else
        - counts[word] = 1
    - return count

Set Default Method

- film_directors = {
  - “The Godfather”: “Francis Ford Coppola”,
  - “The Rock”: “Micael Bay”,
  - “Goodfellas”: “Martin Scorsese”)
- }
- setDefault : search for key and if does not exist add it dictionary along with the second parameter
  - If you do not provide a second parameter then it will add none
- Example
  - film_directories.setDefault(“Bad Boys”, “Michael Bay”) // Adds Key: Bad Boys and value Michael Bay
  - film_directories.setDefault(“Bad Boys”, “Michael Bay”) // Does not add key
  - film_directories.setDefault(“Charles Stockman”) // Adds Key: Charles Stockman and value None

Pop Method

- Pop accepts a key and a value to be returned if the key is not found and return the value
  - If the key does not exist and no default value is provided as the second parameter get a key error exception
- Example
  - release_dates = {
    - “Python” : 1991,
    - “Ruby”: 1995,
    - “Java”: 1995,
    - “Go”: 2007
  - }
  - year = release_dates.pop(“Java”) // Removes the key/value pair if present and returns 1995
  - year = release_dates.pop(“Rust”) // Key Rust does not exist throws KeyValue Exception
  - year = release_dates.pop(“Rust”, 2000) // Returns 2000
- del –> remove key/value pair
  - del release_dates[“Python”] // Does not return corresponding value
    - If key does not exist raise a key/value excpeiton

Clear Method

- Example
  - web_sites = {
    - “Wikipedia”: “http://www.wikipedia.org”,
    - “Google”: “http://www.google.com”
  - }
- websites.clear() // The len(websites) == 0
- del website // Destroys the Object referenced by website
-
  -
  -

Update Method

- Use one dictionary to update another
- Example
  - employee_salaries = {
    - “Guido”: 100000,
    - “James”: 500000,
    - “Brandon”: 900000
  - }
  - extra_employee_salaries = {
    - “Yukihiro”: 1000000
    - “Guido”: 333333
  - }
- employee_salaries.update(extra_employee_salaries) // The extra_employee_salaries are added to employee_salaries
- If we have a duplicate key then the key value pair from the directory passed into update will overwrite the original key.

Dict Function

- print(list(abc)) will return [‘a’, ‘b’, ‘c’]
- Example : employee_titles =[
  - [“Mary,”, “Senior Manager”],
  - [“Brian”, “Vice President”],
  - [“Julie”, “Assistant Vice President”]
-]
- dict(employee_title) produce a dictionary where the names are the keys and the values are the job titles
- A dictionary is datatype form mutable object that store mapping of unique key and values
  -

Iterate over a Directory in a while loop
The Items Method

- Example
  - Chinese_food = {
    - “Sesame Chicken”: 9.99,
    - “Fried Rice”: 1.99
  - }
  - for food in Chinese_food:
    - print(f”The food is {food} and its price is {chinese_food[food]}
- The Items Method
  - college_course = {
    - “history”: “Mr. Washington”,
    - “math”: “Mr. Newton”
  - }
  - for key, value in college_courses.items()
    - print(f”The course {key} is being taught by {value}”)
  - for _ , value in collegecourses.items() // The underscore means the value is not being used
    - print(f”The professor is { value }”)

The Key and Values Methods

- Both methods return an iterable dictionary view object
  - same as the items method
- Example
  - cryptocurrency_price = {
    - “Bitcoin” : 400000,
    - LiteCoin: 10
  - }
  - print(cryptoCurrent_prices.keys()) // dict_keys([‘Bitcoin’, ‘LiteCoin’]) and the class is dict_keys
  - for currency in cryptoCurrency_prices.keys(): // for loop iterating over key
    - pass
  - for price in cryptoCurrency_prices.values() // for loop iterating over values
    - pass
  - “Bitcoin” in cryptoCurrency_keys() // Will return true
- Example – Create and sort a list of the dictionary’s keys where the dictionary name is verse_dict : print(sorted(verse_dict.keys()))

Sorted Function

- The Sort will return a list of sorted dictionary key
- Example
  - salaries = { “Executive Assistant”: 20, “CEO”: 100 }
  - print(sorted(salaries)) // [“CEO”, “Executive Assistant”]

Keyword Arguments (**kwargs)

- A keyword argument is one where we provide the name of a parameter when we invoke the argument
- ** before any parameter will tell python we will expect any number of keyword arguments
  - **kwargs bundle all the keyword args into a dictionary
  - Example
    - def collect_keyword_arguments(**kwargs):
      - print(**kwargs)
    - collect_keyword_arguments(a = 2, b = 3, c = 4)
      - the routine will print { ‘a’ : 2, “b” : 3, “c” : 4 } // This type of kwargs is a dict
  - The *args must always come before the **kwargs
  - Example
    - def args_and_kwargs(a, b, *args, **kwargs)
      - print(f“The total of regular arguments is { a +b }”)
      - print(f”The total of the args arguments is { sum(*args) }”)
      - print(f”The total of the args arguments is { sum(dict_total.values()) }”)
    - print(1,2) → 3
    - print(1,2,3,4,5,6,x=8, y=9, z=10) → 3, 18
    - print(1,2,3,4,5,6,x=8, y=9, z=10) → 3, 18, 27

Keyword Arguments (**kwargs)

- Example
  - def my_func(a, b, *args, **kwargs):
    - print(kwargs)
  - my_func(b = 3, a = 10, c = 4) will produce { c:15 }
  - my_func(20, 30, 40, 50) would produce {} since *args would get 40 and 50

Dictionary: Unpacking Arguments Directory

- example
  - def height_to_meters(feet, inches):
    - total_inches = (feet * 12) + inches
    - return total_inches = .0254
  - stats = {
    - “feet”: 5,
    - “inches”: 11
  - }
  - print height_to_meters(**stats) will make it equal to height_to_meters(5, 11)
  - stats2 = {
    - “feet”: 5.
    - “inches”: 11
    - nonsnese: true
  - print height_to_meters(**stats2) will cause height_meter to throw a type error since there is 3 key/values and the function expects 2 key/values
  -

List Comprehensions Dictionary Comprehensions 1 & 2

- Example
  - languages = [“Python”, “JavaScript”, “Ruby”]
  - lengths = { language: len(language) for language in languages }
  - lengths = { language: len(language) for language in languages if “t” in language }
- Example
  - capitals = {
    - “New York”, “Albany”,
    - “California”: “Sacramento”,
    - “Texas”, “Austin”
    - }
  - inverted = { capital:state for state, capital, capitals.items() if len(state) != len(capital }

Sets: Intro To Sets

- A mutable unordered datastructure that prohibits duplicate values where the order is not guaranteed
- Example
  - stocks = { “MSFT”, “FB”, “IBM”, “MSFT” }
  - print(stocks) { “FB”, “MSFT”, “IBM” }
  - lottery_numbers = { (1,2,3), (4,5,6), (1,2,3) }
  - print(lottery_numbers) { (1,2,3), (4,5,6) }
- A set can only store immutable objects so it cannot store list or dictionaries
- Example
  - len(stocks) The answer would be 3
  - “MSFT” in stocks true
  - print(stocks[2]) returns a TypeError
  - for stock in stocks order of element is not guarrenteed
    - print(stock)
  - squares = { number** 2 for number in [-5, -4 , -3 , 3, 4 , 5] } will return
  - print(squares) {16,25,9}

Sets: Set Function
Add or Update Methods

- Example
  - set() returns an empty set {} which is only way in Python to create an empty set
  - print(set((1,2,2,1,3))) // returns { 1,2,3}
  - print(set("abc")) // { 'a', 'b', 'c' }
  - print(set({ "key": "value" })) // { "key" }
- Example
  - philosophers = ["Plato", "Socrates", "Aristotle", "Pythagoras", "Socrates", "Plato"]
    - philosophers_without_duplicates = list(set(philosophers))
- Add or Update Methods
  - Example
    - disney_character = { "Mickey Mouse", "Minnie Mouse" }
    - disney_character.add("Ariel") { "Mickey Mouse", "Minnie Mouse", "Ariel" }
    - disney_character.update("Mickey Mouse", "Donald Duck") { "Donald Duck", "Minnie Mouse", "Ariel" }
    - disney_character.add("Ariel") { "Donald Duck", "Minnie Mouse", "Ariel" }
  - pages = { 10,20,30 }
  - pages.add({ 30, 40, 50 }) // TypeError. A mutable list cannot be added to a set.

The Remove and Discard Methods

- The Remove raise a keyError if the value is not there. The discard method does not raise the exception
- Example
  - agents = { “Mulder”, “Scully”, “Doggert”, “Reyes” }
  - agents.remove(“Doggert”) // agents = { “Mulder”, “Scully”, “Reyes” }
  - agents.remove(“Skinner”) // keyError
  - agents.remove(“Scully”) // agents = { “Mulder”, “Reyes” }
  - agents.remove(“test”) // agents = { “Mulder”, “Reyes” }
- Example
  - pages = { 10, 20, 30 }
  - element = pages.remove(30) // None
  -

Sets : The Intersection Method to Identify Common Elements Between Sets
The Union Method to combine elements from two sets
The Difference Methods to Identify Element Not in Common Between Two Sets

- candy_bars = { “Milky Way”, “Snickers”, “ 100 Grand” }
- sweet_things = { “Sour Patch Kids“, “Reeses Pieces”, “Snickers” }
- Example
  - print(candy_bars.intersection(sweet_things)) // { “Snickers” }
  - print(candy_bars & sweet_things) // { “Snickers” }
- Union
  - Example
    - print(candy_bars.union(sweet_things)) // { “Milky Way”, “Snickers”, “100 Grand”, “Source Patch Kids”, “Reeses Pieces }
    - print(candy_bars | sweet_things) // { “Milky Way”, “Snickers”, “100 Grand”, “Source Patch Kids”, “Resses Pieces }
- Difference – Found in the Calling Function, but not the Argument List
  - Example
    - print(candy_bars.difference(sweet_things)) // { “Milky Way”, “Reeses Pieces” }
    - print(candy_bars - sweet_things) // { “Milky Way”, “Reeses Pieces” }

The Symmetric difference Method to identify elements not in common between two sets

- Elements found in either set, but not both
- Example
  - candy_bars = { “Milky Way”, “Snickers”, “ 100 Grand” }
  - sweet_things = { “Sour Patch Kids“, “Reeses Pieces”, “Snickers” }
  - print(candy_bars.symmetric_difference(sweet_things)) // { “Milky Way”, “Snickers”, “100 Grand”, “Sour Patch Kids, “Reeses Pieces” }
  - print(candy_bars ^ sweet_things) // { “Milky Way”, “Snickers”, “100 Grand”, “Sour Patch Kids, “Reeses Pieces” }

Set: The issubset and issuperset

- Example
  - a = { 1, 2, 4 }
  - b = { 1, 2, 3, 4, 5 }
  - print(a.issubset(b)) // True
  - print(a < b) // True
  - print(a <= b) // True
  - print(b.issubset(a)) // False
  - print(b.issuperset(a)) // True
  - print(b > a) // True
  - print(b >= a) // True
  - print(a.issuperset(b)) // False
  -

Set: The Frozenset Object

- An immutable set
- Example
  - mr_freeze = frozen_set([1,2,3,2])
  - print(mr_freeze) // frozenset({1,2,3})
  - mr_freeze.add(4) // get an Attribute Error
- Dictionary keys can only be immutable objects. A frozen set can serve as a dictionary key.
- Example
  - regular_set = frozen_set({ 1,2,3 })
  - print({ regular_set: “some_value”})

The Remove and Discard Methods

- The Remove raise a keyError if the value is not there. The discard method does not raise the exception
- Example
  - agents = { “Mulder”, “Scully”, “Doggert”, “Reyes” }
  - agents.remove(“Doggert”) // agents = { “Mulder”, “Scully”, “Reyes” }
  - agents.remove(“Skinner”) // keyError
  - agents.remove(“Scully”) // agents = { “Mulder”, “Reyes” }
  - agents.remove(“test”) // agents = { “Mulder”, “Reyes” }
- Example
  - pages = { 10, 20, 30 }
  - element = pages.remove(30) // None
- pop
  - Removes a random element
  - print(agents.pop()) // prints and removes aa randome elements

Generators

- A simple way to create iterators using functions
- Example
  - def my_range(x):
    - I = 0
    - while I < x:
      - yield I // Allows a function to return a value one at a time and starts at the last value
      - I += 1
- Generators are a lazy way to build iterables.
  - Useful when a fully realized list would not fit into memory
  - cost to calculate element is hight and you want to do it as late as possible
- Can be iterated over only once.
- sq_iterator = (x**2 for x in range(10)) # this produces an iterator of squares // Creates a generator using list comprehension
- Example – Start the first line with the given number and increment it
  - lessons = ["Why Python Programming", "Data Types and Operators", "Control Flow", "Functions", "Scripting"]
  - def my_enumerate(iterable, start=0):
  - for iter in iterable:
  - yield start, iter
  - start += 1
  - for i, lesson in my_enumerate(lessons,6):
  - print("Lesson {}: {}".format(i, lesson))

Summary

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	[ ] or list()	[5.7, 4, 'yes', 5.7]
Tuple	Yes	No	( ) or tuple()	(5.7, 4, 'yes', 5.7)
Set	No	Yes	{}* or set()	{5.7, 4, 'yes'}
Dictionary	No	No**	{ } or dict()	{'Jun': 75, 'Jul': 89}

- * You can use curly braces to define a set like this: {1, 2, 3}. However, if you leave the curly braces empty like this: {} Python will instead create an empty dictionary. So to create an empty set, use
-
- ** A dictionary itself is mutable, but each of its individual keys must be immutable.