**Google It with Python**

**Crash Course on Python**

**Glossary terms from course 1, module 1**

**Terms and definitions from Course 1, Module 1**

**Automation:** The process of replacing a manual step with one that happens automatically

**Client-side scripting language:** Primarily for web programming; the scripts are transferred from a web server to the end-user’s internet browser, then executed in the browser

**Code editors:** Tools to provide features, including syntax highlighting, automatic indentation, error checking, and autocompletion

**Computer program:** A step-by-step list of instructions that a computer follows to reach an intended goal

**Functions:** A reusable block of code that performs a specific task

**IDE:** A software application that provides comprehensive facilities for software development

**Interpreter:** The program that reads and executes code

**Input:** Information that is provided to a program by the end user

**Logic errors:** Errors in code that prevent it from running correctly

**Machine language:** Lowest-level computer language. It communicates directly with computing machines in binary code (ones and zeros)

**Object-oriented programming language:** Most coding elements are considered to be objects with configurable properties

**Output:** the end result of a task performed by a function or computer program

**Platform-specific scripting language:** Language used by system administrators on those specific platforms

**Programming:** The process of writing a program to behave in different ways

**Programming code**: A set of written computer instructions, guided by rules, using a computer programming language

**Programming languages:** Language with syntax and semantics to write computer programs

**Python:** A general purpose programming language

**Python interpreter:** Program that reads and executes Python code by translating Python code into computer instructions

**Script:** Often used to automate specific tasks

**Semantics:** The intended meaning or effect of statements, or collections of words, in both human and computer languages

**Syntax:** The rules for how each statements are constructed in both human and computer languages

**Variables:** These are used to temporarily store changeable values in programming code

Mark as completed

Like

Dislike

Report an issue

**Terms**

* **expression** - a combination of numbers, symbols, or other values that produce a result when evaluated
* **data types** - classes of data (e.g., string, int, float, Boolean, etc.), which include the properties and behaviors of instances of the data type (variables)
* **variable** - an instance of a data type class, represented by a unique name within the code, that stores changeable values of the specific data type
* **implicit conversion** - when the Python interpreter automatically converts one data type to another
* **explicit conversion** - when code is written to manually convert one data type to another using a data type conversion function:
  + **str()** - converts a value (often numeric) to a **string** data type
  + **int()** - converts a value (usually a float) to an **integer** data type
  + **float()** - converts a value (usually an integer) to a **float** data type

**Tuple use cases**

Remember, there are a number of cases where using a tuple might be more suitable than other data types:

* Protecting data: Because tuples are immutable, they can be used in situations where you want to ensure the data you have cannot be changed. This can be particularly helpful when dealing with sensitive or important information that should remain constant throughout the execution of a program.
* Hashable keys: Because they're immutable, tuples can be used as keys on dictionaries, which can be useful for complex keys.
* Efficiency: Tuples are generally more memory-efficient than lists, making them advantageous when dealing with large datasets.

**List comprehensions**

A list comprehension is an efficient method for creating a new list from a sequence or a range in a single line of code. It is a best practice to add descriptive comments about any list comprehensions used in your code, as their purpose can be difficult to interpret by other coders.

* **[expression for variable in sequence]** - Creates a new list based on the given sequence. Each element in the new list is the result of the given expression.
  + Example: **my\_list = [ x\*2 for x in range(1,11) ]**
* **[expression for variable in sequence if condition]** - Creates a new list based on a specified sequence. Each element is the result of the given expression; elements are added only if the specified condition is true.
  + Example: **my\_list = [ x for x in range(1,101) if x % 10 == 0 ]**

Note that tuples do not have comprehensions but a similar functionality can be achieved with:

**tuple(i for i in (1, 2, 3))**

**When to use for loops or list comprehensions**

In Python, list comprehensions are generally used for creating new lists from existing ones in a concise and readable manner, especially when the task involves simple transformations or filtering of elements.

**for** loops are more versatile and are preferred when the operation is more complex, requires multiple lines of code, involves statements other than expression (like **print**, **pass**, **continue**, **break**), or when you need to iterate over a list without creating a new one.

Notable example :

def biography\_list(people):

    # Iterate over each "person" in the given "people" list of tuples.

    for  person in people:

        # Separate the 3 items in each tuple into 3 variables:

        # "name", "age", and "profession"

        name,age,profession=people

        # Format the required sentence and place the 3 variables

        # in the correct placeholders using the .format() method.

        print("{} is {} year old and work as a ".format(name ,age,profession))

# Call to the function:

biography\_list([("Ira", 30, "a Chef"), ("Raj", 35, "a Lawyer"), ("Maria", 25, "an Engineer")])

# Click Run to submit code

# Output should match:

# Ira is 30 years old and works as a Chef.

# Raj is 35 years old and works as a Lawyer.

# Maria is 25 years old and works as an Engineer.

**Key takeaways**

**while** loops and **if else** statements are methods you can use to search for values within a dictionary. If you want to search through an entire dictionary quickly to find a value, use a **while** loop. If you want to search through a dictionary to find and check a value, use an **if else** statement to return the value and perform specific actions based on the result.

**Study Guide: Dictionary Methods**

Python dictionaries are used to organize elements into collections. Dictionaries include one or more keys, with one or more values associated with each key**.**

**Operations**

* **len(dictionary)** - Returns the number of items in a dictionary.
* **for key in dictionary** - Iterates over each key in a dictionary.
* **for key, value in dictionary.items()** - Iterates over each key,value pair in a dictionary.
* **if key in dictionary** - Checks whether a key is in a dictionary.
* **dictionary[key]** - Accesses a value using the associated key from a dictionary.
* **dictionary[key] = value** - Sets a value associated with a key.
* **del dictionary[key]** - Removes a value using the associated key from a dictionary.

**Methods**

* **dictionary.get(key, default)** - Returns the value corresponding to a key, or the default value if the specified key is not present.
* **dictionary.keys()** - Returns a sequence containing the keys in a dictionary.
* **dictionary.values()** - Returns a sequence containing the values in a dictionary.
* **dictionary[key].append(value)** - Appends a new value for an existing key.
* **dictionary.update(other\_dictionary)** - Updates a dictionary with the items from another dictionary. Existing entries are updated; new entries are added.
* **dictionary.clear()** - Deletes all items from a dictionary.
* **dictionary.copy()** - Makes a copy of a dictionary.

**Dictionaries versus Lists**

Dictionaries are similar to lists, but there are a few differences:

**Both dictionaries and lists:**

* are used to organize elements into collections;
* are used to initialize a new dictionary or list, use empty brackets;
* can iterate through the items or elements in the collection; and
* can use a variety of methods and operations to create and change the collections, like removing and inserting items or elements.

**Dictionaries only:**

* are unordered sets;
* have keys that can be a variety of data types, including strings, integers, floats, tuples;.
* can access dictionary values by keys;
* use square brackets inside curly brackets { [ ] };
* use colons between the key and the value(s);
* use commas to separate each key group and each value within a key group;
* make it quicker and easier for a Python interpreter to find specific elements, as compared to a list.

**Lists only:**

* are ordered sets;
* access list elements by index positions;
* require that these indices be integers;
* use square brackets [ ];
* use commas to separate each list element

**Glossary terms from course 1, module 4**

**Terms and definitions from Course 1, Module 4**

**Dictionaries:** A data type used to organize elements into collections, taking the form of pairs of keys and values

**List comprehensions:** Create new lists based on sequences or ranges

**String:** A data type used to represent a piece of text. sequences of characters and are immutable

**Tuples:** Sequences of elements of any type that are immutable, written parentheses instead of square brackets