**Comments:** demarcated by the use of the pound sign/hashtag (# or triple quotes (‘ ‘ ‘….’ ‘ ‘ or “””…”””)

**Variable:** names given to data that we need to store and manipulate in the programs

* Every time you declare a new variable, you need to give it an initial value, but that can be changed later
* A variable name can only contain letters, numbers, or underscores
* The first character of a variable cannot be a number
* Reserved words cannot be used as a variable name, such as print, input, if, else, etc.
* Variable names are case sensitive (username DNE username)
* Multiple variables can be declared at once
  + userAge, userName = 30, ‘Peter’
* Two variable naming conventions in Python
  + Camel case notation: writing compound words with mixed casing (userName)
  + Underscores to separate words (user\_name)

**Operators:**

* **Assignment operator (=):** the = is the assignment operator. It means that we are assigning the *value* on the right side of the = sign to the *variable* on the left
  + x = y DNE y = x
  + x = 5, y = 10, x = y
    - print (“x = “, x) and print (“y = “, y) produces
      * x = 10, y = 10
  + x = 5, y = 5, x = y
    - print (“x = “, x) and print (“y = “, y) produces
      * x = 5, y = 5
* **Mathematical operators**
  + **Addition (+)**
  + **Subtraction (-)**
  + **Multiplication (\*)**
  + **Division (/)**
  + **Floor division(//)** rounds down the answer to the nearest whole number
  + **Modulus (%)** returns the remainder of a division equation
  + **Exponent (\*\*)** power symbol, can use negative numbers for square root, etc.
* **Other assignment operators**
  + x = 5, then x = x + 2 produces
    - x = 7 because the program first evaluates the expression on the right
    - the assignment operator **(+=)** does the same thing
      * x += 2 is the same as x = x + 2
  + x -= 2 is the same a x = x – 2
  + x \*= 2 is the same as x = x \* 2
  + x /= 2 is the same as x = x / 2
  + x //= 2 is the same as x = x // 2
  + x %= 2 is the same as x = x % 2
  + x \*\*= 2 is the same as x = x \*\* 2

**Data Types**

* **Integers** are numbers with no decimal parts, from negative infinite to positive infinite
* **Float** refers to numbers that have a decimal part, such as 1.234
* **String** refers to text
  + To declare a string, the value must be enclosed in single or double quotes
  + Numbers, when enclosed in single or double quotes, are stored as data type string instead of integers or floats
  + Strings can be concatenated/combined by using the concatenate sign **(+)**
    - “Peter” + “Lee” is equivalent to “Peter Lee”
  + String-type data have built-in functions
  + Strings can also be formatted using the **(%)** operator. The syntax is:
    - “String to be formatted” %(values or variables to be inserted into the string, separated by commas)
    - “String in quotes” + the % symbol + a pair of parentheses in which we write the values or variables to be inserted into the string (called a tuple)
    - Example:
      * brand = “Apple”

exchangeRate = 1.235235245

message = “The price of this %s laptop is %d USD and the exchange rate is %4.2f USD to 1 EUR” %(brand, 1299, exchangeRate)”

print (message)

output: The price of this Apple laptop is 1299 USD and the exchange rate is 1.24 USD to 1 EUR

* + - * %s, %d, and %4.2f are **formatters**; they serve as placeholders in the string
        + %s formatter is used to represent a string
        + %d formatter represents an integer

If we want to add spaces before an integer, we can add a number between % and d to indicate the desired length of the string: %5d%(123) will give us “ 123” with two spaces in front and a total length of 5

* + - * + %f formatter is used to format floats. In %4.2f, 4 refers to the total length and 2 refers to 2 decimal places. To add 3 spaces before 1299, we can format this ass %7.2f
  + The format **( )** method can also be used to format strings. The syntax is
    - “string to be formatted”.format(values or variables to be inserted into string, separated by commas)
    - When using the format ( ) method, don’t use %s, %f, or %d as placeholders
    - When using the format( ) method, use curly braces **{ }** as palceholders
      * Example: “The price of this {0:s} laptop is {1:d} USD and the exchange rate is {2:4.2f} USD to 1 EUR”.format(“Apple”, 1299, 1.235234245)

Output: The price of this Apple laptop is 1299 USD and the exchange rate is 1.24 USD to 1 EUR

* + - * Inside the braces, write the position of the argument to use (0,1,2 here, like index position), followed by a colon. After the colon, write the formatter.
      * There should be no spaces within the curly braces
      * If you do not want to format the string, you can write: “The price of this {} laptop is {} USD and the exchange rate is {} USD to 1 EUR”.format(“Apple”, 1299, 1.235234245).
        + Here, the Python interpreter will replace the braces based on the order of the arguments provided.
        + Output: The price of this Apple laptop is 1299 USD and the exchange rate is 1.235235245 USD to 1 EUR
* Data types can be changed in three different ways
  + The **int()** function takes in a float or an appropriate (numerical) string and converts it to an integer
    - int(5.712987) will result in 5
    - int(“4”) will result in 4
    - int(“Hello”) or int(“4.22321”) will result in error
  + The **float()** function takes in an integer or an appropriate string and turns it into a float
    - float(2) or float(“2”) will result in 2.0
    - float(“2.09109”) will result in 2.09109
  + The **str()** function converts an integer or a float to a string
    - str(2.1) will result in “2.1”
* **List** refers to a collection of data which are normally related
  + To declare a list, you write listName = [initial values] –use square brackets. Multiple values are separated by commas.
    - Example: userAge = [21, 22, 23, 24, 25]
  + Lists can also be assigned without any initial value
    - Example: listName = [ ]
    - Items can be added to the list using the .append() method.
    - Individual values in the list are accessible by their indexes, starting from 0
    - You can also access the values of a list from the back, which begins with an index of -1 for the last position
    - You can assign a list, or part of it, to a variable
      * If you write userAge2 = userAge, then userAge2 = [21, 22, 23, 24, 25]
      * If you write userAge3 = userAge[2:4], then userAge3 = [23,24]
    - The notation 2:4 in the above example is known as a **slice*.*** Whenever we use the slice notation in Python, the item at the start of the index is always included, but the item at the end is always excluded.
      * The slice notation includes a third number known as the **stepper.** If we write userAge4 = userAge[1:5:2], we will get a sublist consisting of every second number from index 1 to index 5-1 because the stepper is 2. So userAge4 = [22, 24]
      * Slice notation have defaults. The default for the first number is 0, and the default for the second number is the size of the list being sliced. This means that if you are starting the index from 0, you can leave the first number placeholder blank, i.e. [ :4].
      * In the above userAge list, [ : ] is the same as [0: ] and [ :5] and [0:5]
  + To modify items in a list, we write **listName[index of item to be modified] = new value**.
    - In the userAge example, if you want to modify the second item, you write userAge[1] = 5, and your list becomes userAge = [21, 5, 23, 24, 25]
  + To add items, you use the **.append()** function.
    - If you write userAge.append(99), your list is now userAge = [21, 5, 23, 24, 25, 99]
  + To remove items, use **del listName[index of item to be deleted].** 
    - If you write del userAge[2], your list is now userAge = [21, 5, 24, 25, 99]
* **Tuples** are like list, but you cannot modify their values. The initial values are the values that will stay for the rest of the program. Tuples are useful when your program needs to store the names of the months of the year, for example.
  + To declare a typle, you write **tupleName = (initial values)** –uses parentheses to declare tuples instead of brackets. Multiple values are separated by a comma.
    - Example: monthsOfYear = (“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, “Dec”)
  + You access the individual values of a tuple using their indexes, like with a list.
    - Example: monthsOfYear[0] = “Jan” & monthsOfYear[-1] = “Dec”
* **Dictionary** is a collection of related data pairs.
  + To write a dictionary, you write **dictionaryName = {dictionary key:value}** –use curly braces and colon. Multiple pairs are separated by commas.
  + Dictionary keys must be unique within one dictionary.
    - Example: myDictionary = {“Peter”:38, “John”:51, “Peter”:13} is no good
    - Example: userNameAndAge = {“Peter”:38, “John”:51, “Alex”:13, “Alvin”:”Not Available”} is good
  + You can also declare a dictionary using the **dict()** method. You would use parenthesis instead of curly braces and you do not put quotation marks for the dictionary keys.
    - In the example above, you would write userNameAndAge = dict(Peter = 38, John = 51, Alex = 13, Alvin = “Not Available”)
  + To access individual items in the dictionary, we use the dictionary key, which is the first value in the pair, instead of the index position like for lists and tuples.
    - Example: To get John’s age, you would write userNameAndAge[“John”] to get the value of 51.
  + To modify items in a dictionary, write **dictionaryName[dictionary key to be modified] = new value**
    - Example: To make John’s new age 21, we write userNameAndAge[“John”] = 21
  + To declare a dictionary without assigning any initial values to it, we simply write dictionaryName = { }
  + To add items to a dictionary, write **dictionaryName[dictionary key] = value**
    - Example: if we want to add Joe, aged 40, we write userNameAndAge[“Joe”] = 40 and it will append “Joe”:40 to the end of the dictionary.
  + To remove items from a dictionary, we write **del dictionaryName[dictionary key]**
    - Example: To remove the “Alex”:13 pair, we write del userNameAndAge[“Alex”]

**Functions** are blocks of reusable code that performs a certain task

**For Loop**

**We are telling python w want to run a specific piece of code multiple times—called iteration.**

**For loops uses definitive iteration, which means we have to specify how many times we want our loops to run. We need to give it an iterable.**

**The number of items on a list will tell the function how many times to run.**

**The for loop syntax is—the use of “for… in ()” and indenting the next line**

**The range() iterable is good to use if you don’t have a list. You can just pass in numbers inside the ()**

**The enumerate() function**