Compiled Coding Level 1- 1-5

INK AND METAL 2020-2021

Concepts Covered in This Course

General Learning Process

- For each concept learned, there will be interactive challenge problems presented to each student.
- At the end of each class, homework projects incorporating all of the lessons for the day will be assigned.
    - This homework is expected to be completed by the start of the next class.

Data Types

Variables

- Understanding variables
- Declaring variables
    - Variable naming conventions
- Assigning values to variables

Integers and Floats

- Declaration of integers and floats
- Understanding the difference between integers and floats
- Arithmetic Operators
    - Addition
    - Subtraction

    - Multiplication
    - Division (including integer division)
    - Exponents
    - Modulo

Strings

- Understanding what strings are
- Declaration of strings
- Concatenating strings
- String functions
    - Displaying strings to the user
    - Finding the length of strings
    - Returning the uppercase values of strings
    - Returning the lowercase values of strings
- Receiving user string input
- Converting strings to integers and vice-versa

Lists, Tuples, and Dictionaries

- Declaration of lists, tuples, and dictionaries
- Understanding the difference between lists, tuples, and dictionaries
- Searching lists and tuples through index
- Slicing lists and tuples
- List and tuple methods
    - Appending to lists and tuples
    - Inserting items into lists and tuples
    - Removing items from lists and tuples
    - Sorting lists and tuples
- Separating Strings into lists (Helpful for reading CSV files when interpreting data)
- Accessing items in a dictionary by key
- Adding items to a dictionary
- Removing items from a dictionary
- Nesting lists, tuples, and dictionaries

If / Elif / Else Statements

- How If / Elif / Else Statements work
- Comparison Operators
    - Greater than
    - Less than
    - Equal to
    - Not equal to
    - Greater or equal than
    - Less than or equal than
- Logical Operators
    - And
    - Or
    - Not
- Booleans
    - Definition
    - Implementation

Loops

- Understanding the differences between while and for loops
    - Iteration of lists, tuples, and dictionaries with for loops
- Declaring while and for loops
- Loop control statements
    - Break
    - Continue
    - Pass

Exceptions

- Understanding exceptions
- Handling exceptions
- Raising exceptions

- Defining clean up actions

Functions

- What is a function?
    - Arguments
- Understanding the scope of variables
    - Global variables
    - Local variables
    - Nonlocal variables
- Defining a function
    - Adding parameters to a function
- Returning an output
    - Implementing this output in the remainder of the program
- Calling/using a function
- Using import
    - Sample packages
    - Creating and using modules
- Understanding and implementing recursion
    - Base case
    - Termination conditions
- __init__

Object Oriented Programming

- Understanding object oriented programming
- Creating classes
    - Constructors
    - Objects
        - Attributes
        - Methods
        - Access modifiers
            - Public

        - Private
        - Protected
- Understanding inheritance
    - Child (derived) vs parent (base) classes
    - How attributes and methods are passed through the classes
    - Method overriding
    - Using super constructors
- Understanding encapsulation and implementing it
- Understanding polymorphism and implementing it
- Understanding multiple inheritance, multilevel inheritance, and method resolution order
- Using operator overloading

Python 3 Programming Course 1

Introduction to Python Programming Environment

Teaching - Shell vs Interpreter

- Interpreter: The interpreter executes the Python source code.
- Shell: The Python shell is an interactive line-by-line way of executing Python code.

Teaching - Comments

- Comments: is not read or executed
    - Usually used for explaining what is happening in the code (Ex: if someone else will read your code later or if you need to document your code)

- Use a # if it's only one line and "" on either side if it's more than one line

```
"""
This is a comment
written in
more than just one line
"""
# this is a comment in one line
print("Hello, World!")
```

Introduction to Integers

Teaching - What is a data type?

- The type of a value (Ex: number, character, etc.) except in python there are specific categories, each has its own functions

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

-

Teaching - What is an integer and float?

- In mathematics, an integer is defined as a whole number that can be positive, negative, or 0.

- Python integers are data types which can hold mathematical integers.
    - REMEMBER: Python integers cannot have decimal places!
- If you want to store a number with decimals in Python, the float data type should be used. A float is an integer with a decimal component.

Challenge Problems

Older Students

Question: Can anyone give me an example of an integer?
Answer: [Accept any answer of a positive or negative number which does not have any decimals (ex. -15)]

Question: Can anyone give me an example of a float?
Answer: [Accept any answer of a positive or negative number with decimals (ex. 3.14)]

Younger Students

Question: Is 3 an integer?
Answer: Yes.

Question: Is -7 an integer?
Answer: Yes.

Question: How about -6.5?
Answer: No.

Question: Then what is it?
Answer: A float.

Teaching - Arithmetic with integers

- Use +, -, *, **, / (opt: %)

- Notice: 16/5 = 3.2 Why? Python auto corrects one number to a float
    - If you want to return an int: use double forward slashes (Ex:16//5 gives you 3)
- PEMDAS: any operation within parentheses will come first, followed by exponents, multiplication/division (left to right), then addition/subtraction (left to right)

```
>>>2 + 3
5

>>>5 - 4
1

>>>3 * 8
24

>>>16 / 5
3.2

>>>16 // 5
3

>>>16 % 5
1

>>>(4 + 3) * 2
14
```

Challenge Problem

Older students

Question: Use python to find x: (5^7 + 201,564) * 2/7 = x
Answer: 319,644.57142

Younger Students

Question: Use python to find the answer to 45 times 1067 divided by 14 minus 2384
Answer: 1,045.64285714

Teaching - print() statement

- From your code, to output, we can use a function called print()

```
print(4)     #Will output the string ‘4’

print(4+5)   #Will output the string ‘9’
```

- For now, just put integers as what you want to print

Challenge Problem

Use python to display the integer result of 2 + 2
Answer: print(2 + 2)

Use python to display the float result of

Teaching - Variables with integers

- Def: variables are just a container to store a value
    - Ex: x = 4 then we have a container called x with value 4
- In Python, the value can be different data types (integer, float, string)
    - Do not need to declare its type like java, and the type can be changed if you redefine the variable
- Declaring a Variable
    - Variable_name = value
- Syntax issues with variable names:
    - CANNOT start with a number

    - MUST start with a letter or _
    - Can contain ONLY a-z, 0-9, _
    - Are CASE-SENSITIVE

```
x = 5
y = 4.3
```

- Things you can do with variables
    - https://www.w3schools.com/python/python_variables.asp
    - The link above has lots of examples in case they want more info

```
x = 2
y = x          #y now equals 2

x = x + 4
print(y)       #y equals 2
print(x)       #x equals 6

z = x + y      #z equals 8

y = 4.3        #y is now of type float instead of int

del x          #x no longer exists
```

Challenge Problem

Question: Store the integers 3 and 2 in variables x and y. Then store the result of x divided by y as z, and display z.
Answer:

```
x = 3
y = 2

z = x / y
```

```
print(z)      # outputs: 1.5
```

Introduction to Strings

Teaching - Overview on Strings

- Strings are sequences of character enclosed in single or double quotes
- Why single or double?

```
>>> type(“Hello”)
str

>>> type(‘Hello’)
str

>>>type(“I can’t do this”)
str

>>>type(‘I can’t do this”)
syntax error
```

- type() helps identify what kind of datatype the code inside () is
- When using strings, whatever you print out, that is a string, it will be printed as is
- All of the examples in the table are strings because they are surrounded by “”, or ‘’
    - For the last example, you can see that it has both single and double quotes. It’s still a string because

Python looks at the very outer “”s or ‘’s and uses that as the starting identification for a string.

- Syntax error = Python cannot understand your line of code
- “/n” in the middle of string means whatever is after the backslash is on a different line.
    - Must come after a space though

```
>>> print(“Hello \n How are you?”)
Hello
    How are you?
```

- **Characters** = anything you can type on keyboard with one keystroke
    - Letters, numbers, backlash

```
>>> type(“1”)
>>> str

>>> type(“hello there”)
str

>>> type(“/”)
str
```

Challenge Problem

Question: Give an example of something that is a string.
Answer: Accept anything in “” or ‘’

Question: Give an example of something that is not a string.
Answer: Accept anything not in “” or ‘’

Teaching - Using Functions with Strings

- **What are functions? (older students)**

    - Blocks of code that only run when you call them
    - You can pass data into them known as parameters
- **Concatenation** = joining strings together end to end to create a new string

```
>>> print(“Hello.” + “ My name is Vishwa.”)
Hello. My name is Vishwa.

>>> print(“1” + “2”)
12
```

- “+” is within the string but is not enclosed in any quotations, representing the addition symbol
- “Hello.” is String 1 and “ My name is Vishwa.” is string 2
    - The space after the quotations is important

```
>>> print(“Hello.” + “My name is Vishwa.”
Hello.My name is Vishwa.

>>> print(“2” + “ 1”)
2 1
```

- There was no space between the starting quotation mark so there was no space after the “Hello.”
- There was space in second print statement so there was a space between “2” and “1” so it became 2 1
- **Finding the length of string** = len()
    - The string is within ()
    - Tells you how many characters are within the string excluding your starting single or double quotes, and additional spacing

```
>>> len(“Hey there!”)
10
```

```
>>> len(“1,2,3,4”)
7
```

- **Make String Uppercase** = .upper()
    - String goes before “.upper()”
    - All ***letters*** in string become uppercase
        - If they are already uppercase,they remain the same - only works for lowercase letters

```
>>> x = “Hello”
>>> x.upper()
HELLO

>>> “Hello”.upper()
HELLO
```

- x is a variable -- stores the value of something
- “x.upper()” is saying take my variable “x” and make it completely uppercase and return it
- “Hello”.upper() is replacing the variable “x” with “Hello” and executing the same thing as mentioned above
- **How to lowercase** = .lower()
    - String comes before “.lower()”
    - All letters become lowercase
        - Stay the same if they are already lowercase

```
>>> x = “HELLo”
>>> x.lower()
hello

>>> “HELLo”.lower()
hello
```

- x is a variable -- stores a value

- “x.lower()” is saying take my variable “x” and lowercase it completely and then return it
- “HELLo”.lower() is replacing “x” with “HELLo” and doing the same as mentioned above
- **How to input** = input()
    - input() tells the user to enter a value or string that is often stored in a variable
    - Whatever you want the user to input goes in ()

```
name1 = input(‘Name: ‘)
print(name1)
----------------------------OUTPUT-------------------------------
Name: Vishwa
Vishwa

object1 = input()
print(object1)
----------------------------OUTPUT-------------------------------
Stick
Stick

userinput = input(“Enter a number: ”)
userinput = int(userinput) * 5
print(userinput)
----------------------------OUTPUT-------------------------------
Enter a number: 2
10
```

- “name1” is a variable
    - (‘Name: ‘) means that whatever your user inputs, it will be after the “:”
        - This will allow them to specifically respond to your question or desire input
-
    - print(name1) tells Python to print whatever the user’s input was

- “object1” is a variable
    - input() tells the user to input something, and that gets stored to “object1”
    - “print(object1)” tells Python to print the user’s input
- userinput is a variable
        - int() of user input tells Python to convert whatever the user inputs into an integer and then multiply the converted input to 5
            - Everything that a user inputs is stored as a string default
        - “print(user input)” tells Python to return the result

Challenge Problem

Question: Ask the user for his/her age, add 5 to the variable in which it is stored in, and display it.
Answer:

```
age = input(“What is your age? ”)
age = int(age) + 5

print(age)
```

Homework Assignments

1: Triangle Area Calculator from User Input

Prompt: Create a calculator which finds the area of a triangle given input from the user

```
#PSEUDOCODE

Display “Triangle Calculator”

Ask the user for the height of the triangle
Ask the user for the base length of the triangle

Multiply height by base length and then divide by 2

Display the result to the user
```

2: 5-Word Sentence Modifier

Prompt: Ask the user to enter in 5 seperate words; join them together to create a sentence; and display the sentence, the length of the sentence, the fully uppercase sentence, and the fully lowercase sentence

```
#PSEUDOCODE

Ask the user for the first word in the sentence (capitalized)
Ask the user for the second word in the sentence
Ask the user for the third word in the sentence
Ask the user for the fourth word in the sentence
Ask the user for the fifth word in the sentence

Combine these words to form a sentence (with . at the end)

Display the sentence
Display the length of the sentence
Display the fully uppercase sentence
Display the fully lowercase sentence
```

Python 3 Programming Course 2

Booleans

Teaching - Booleans

- Can only have True or False (notice the capital letters)

```
x = True
y = False
```

Teaching - Comparison Operators

- Use these operators to find whether something is True or False

Operator	Meaning	Example
==	Equal to	a==b
!=	Not equal to	a!=b
>	Greater than	a>b
<	Less than	a<b
>=	Greater than or equal to	a>=b
<=	Less than or equal to	a<=b

Challenge Problems

Question

- Create variables x, y, and z and set them to 14, 5, and 5 respectively
- Print whether it is true or false that x is less than or equal to y
- Print whether it is true or false that y is equal to z

Answer

```
x = 14
y = 5
z = 5

print(x<=y)
#will print false

print(y==z)
#will print true
```

Teaching - Logical Operators

- Combine multiple conditional statements together

Operator	Meaning	Example
and	Only returns True if both statements are True	a==b and x==y
or	Returns True if one of the conditions are True	a>b or x<y

not	Gives the opposite result / reverses it	not(a==b and x<=y)

Challenge Problems

Question

- Set variables x, y, and z to 2, 4, 2
- Check whether x is equal to both y and z
- Check whether x is less than y or x is equal to z
- Print false if x is equal to z

Answer

```
x=2
y=4
z=2

print(x==y and x==z)
#will print false

print(x<y or x==z)
#will print true

print(not(x==z))
#will print false
```

If/ Elif / Else Statements

Teaching - What are if statements?

- An if statement is a statement which only runs its code if its boolean condition is satisfied.
- The structure of an if statement looks like this:

```
if <<condition>>:

    <<code>>
```

- Here is an example of an if statement which only prints “three” if the variable x is equal to 3:

```
if x == 3:

    print(“three”)
```

- If the variable x was defined above with the value 3, the print statement would be executed

Challenge Problem

Question

- Create the variable student_number and give it the value of 15
- Create an if statement which checks if student_number is greater than 50
    - If student_number is greater than 50, print “This class is too big!”
- What is the output?

Answer

```
student_number = 15

if student_number > 50:

    print(“This class is too big!”) #No output
```

Teaching - What are else statements?

- If the boolean condition is not satisfied in an if statement, the else statement is run
- The structure of an if and else statement looks like this:

```
if <<condition>>:

    <<code>>

else:

    <<code>>
```

- Here is an example of an else statement which runs if x is not equal to 17:

```
x = 17

if x == 17:

    print(“17”)

else:

    print(“not 17”)
```

- If x is equal to 13, “not 17” would be printed

Challenge Problem

Question

- Create an if statement where if x is equal to 7 and if y is equal to 5, print “true”
- Else print “false”
- Set x to 7 and y to 4

- What is the result?

Answer

```
x = 7
y = 4

if x == 7 and y == 5:

    print(“true”)

else:

    print(“false”) #answer
```

Teaching - What are elif statements?

- Elif is short for “else if”
- An elif statement allows you to check for multiple expressions which evaluate to true (essentially an extended if statement).
- The structure of an elif statement looks like this:

```
if <<condition>>:

    <<code>>

elif <<condition>>:

    <<code>>

else:

    <<code>>
```

- Here is an example of an elif statement:

```
x = 0

if x < 0:

    print(“x is a negative integer.”)

elif x == 0:

    print(“x is equal to 0.”)

else:

    print(“x is a positive integer.)
```

- Since x is not less than zero, the if statement does not run.
- However, since x has the value of zero, the elif condition is true and the statement “x is equal to 0” is printed.

Challenge Problem

Question

- Create an if statement which checks for the length of a string:
    - If a string is 3 characters or less, print: that is a short string
    - If a string is between 5 and 8 characters (inclusive), print: that is an average-sized string
    - Else, print: that is a long string
    - Pass in the string “hello” as the variable my_string and see what the result is

Answer

```
my_string = “hello”

if len(my_string) <= 3:

    print(“that is a short string”)

elif len(my_string) >= 5 and len(my_string) <= 8:

    print(“that is an average-sized string”)

else:

    print(“that is a long string”)
```

Lists

Teaching - What is a list and how to declare it?

- An ordered collection of objects that can be changed
    - Can be numbers, letters, or words
- Declaring a list:
    - Use brackets to enclose the list, and commas to separate each object
    - Uses a number index

```
>>> my_list = [“hi”, “hello”, “hey”]

>>> print(my_list[0])

hi

#same thing applies to call the other items in a list

>>> print(my_list[-1])
```

```
hey

>>> print(my_list[-2])

hello

#same thing applies to call the other items in a list
```

- I define a list called “my_list” and assign items to it
- list.append() is telling Python to get the item that is associated with its number
    - index(0) = the first item in the list
    - index(-1) = last item in the list

- **Adding Lists**

```
>>> my_list = [1,2,3]

>>> another_list = [4,5,6]

>>> new_list = my_list + another_list

>>> print(new_list)

[1,2,3,4,5,6]
```

- I defined a list called “my_list”
- I defined another class called “another-list”
- I defined a new list called “new_list”
- When I print “new_list” which adds both the other lists, you will get a combined version of both those lists into one list

Slicing Lists: Useful if you want to access particular items from your list

```
>>> my_list = [1,2,3,5,6]

>>> print(my_list[0:])

1,2,3,5,6

>>> my_list = [1,2,3,5,6]

>>> print(my_list[: 3])

1,2,3

>>> my_list = [1,2,3,5,6,7]

>>> print(my_list[0:5:2])

1,3,6
```

- I defined a list called “my_list”
    - When I use “[]”, it tells Python that I’m asking for a particular value within whatever I defined
    - If I start with “:” that tells Python to use the first value in my list
        - When I use “:5” it tells python to use my number index and start from the starting value until the number index of 5
        - When I use “ 0: 5: 2” it tells Python to start from the item at index 0 and go to the item with an index of 5 by a step size of 2
            - Similar to going from 1-10 but only saying even numbers

Strings to Lists:

- Use the “list()” method
    - Type your string between “()”

    - This takes a string (remember string must be surrounded by “”)
    - Then splits the string character-wise
- Use the “”.split() method to convert string to a list with words
    - You type your string before the “.split()”
    - Then the “.split()” splits the string into words and return them in the form of a list

```
>>> list(“Hello”)

['H', 'e', 'l', 'l', 'o']

>>> list(“Hey”)

[‘H’, ‘e’, ‘y’]

>>> “Hello World”.split()

[‘Hello’, ‘World’]

>>> “Hi there Vishwa. My name is Jarvis”.split()

['Hi', 'there', 'Vishwa.', 'My', 'name', 'is', 'Jarvis']
```

List Methods

```
>>> my_list = [1,2,4,5,6]

>>> my_list.append(7)

>>> print(my_list)

[1, 2, 4, 5, 6, 7]
```

- I defined a list called “my_list”
- “list.append(7)” adds 7 to the end of the list

```
>>> my_list = [1,2,4,5,6]

>>> my_list.insert(3,7)

>>> print(my_list)

[1, 2, 4, 7, 5, 6]

>>> my_list = [‘Hello’, ‘how’, ‘you’]

>>> my_list.insert(2, ‘are’)

>>> print(my_list)

['Hello', 'how', 'are', 'you']

>>> my_list = [1, 2, 3, 4, 5, 6]

>>> my_list.remove(4)

>>> print(my_list)

[1, 2, 3, 5, 6]

>>> my_list = [‘Ford’, ‘Bugatti’, ‘Ferrari’]

>>> my_list.sort()

>>> print(my_list)

[‘Bugatti’, ‘Ferrari’, ‘Ford’]

>>> my_list = [1, 3, 16764, 45, 8]

>>> my_list.sort()
```

```
>>> print(my_list)

[1, 3, 8, 45, 16764]

>>> my_list = [1, 3, 16764, 45, 8]

>>> my_list.reversal()

>>> print(my_list)

[8, 45, 16764, 3, 1]

>>> my_list = [1, 3, 5, 6]

>>> my_list.pop()

>>> print(my_list)

[1, 3, 5]
```

- I defined a list called “my_list”
- “my_list.insert(3, 7)” tells Python to insert ‘7’ at the index of 3
- Then I told to python to print out the updated version of the list

- I defined a list called “my_list”
- “my_list.insert(2, ‘are’) tells Python to insert ‘are’ at the index of 2
- Then I told to python to print out the updated version of the list

- I defined a list called “my_list”
- “my_list.remove(4) tells Python to search for the number 4 and once it finds it, to remove it from the list

- Then I told to python to print out the updated version of the list

- I defined a list called “my_list”
- “my_list.sort() tells Python to look over all the items in the list and arrange them alphabetically
- Then I told to python to print out the updated version of the list

- I defined a list called “my_list”
- “my_list.sort() tells Python to look over all the items in the list and arrange them from least to greatest
- Then I told to python to print out the updated version of the list

- I defined a list called “my_list”
- “my_list.reverse() tells Python to look over all the items in the list and reverse their order
- Then I told to python to print out the updated version of the list

- I defined a list called “my_list”
- “my_list.pop() tells Python to pop or delete the last item of a list
- Then I told to python to print out the updated version of the list

Challenge Problems

1. Ask the user for a list containing numbers and words
2. Then delete the last item of the list
3. Add another item instead of the deleted one
4. Lastly, insert “accomplished” as the second item in the list

```
        a. Remember indexing
    5. Then reverse the list
    6. Store the changed list into “new_list” and print that out
```

```
>>> h = input("Enter a list: ")

>>> y = h.split()

>>> y.append("hi")

>>> y.pop()

>>> y.insert(1, "accomplished")

>>> y.reverse()

>>> print(y)
```

Making a Nested List

```
>>> L = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

>>> print(L[2])

['cc', 'dd', ['eee', 'fff']]

>>> print(L[2][2])

['eee', 'fff']

>>> print(L[2][2][0])

eee
```

- I defined a list that contained other lists within it
- When I “print(L[2]), I’m telling Python to print the item with the index of 2, and in this case, there is another list with the index of 2; this is totally possible and allowed
- When I “print(L[2][2]), I’m telling Python to print the item from the above reasoning’s item with the index of 2
- When I print(L[2][2][0]), I’m telling Python to print the item from the above reasoning, and then print the item with an index of 0 in it
- You will start to understand it better once you guys practice and see more examples

Dictionaries

Teaching - What is a key and value? What is a dictionary?

- A key corresponds to a value
    - Ex: job is scientist; key is job and value is scientist
- A dictionary is made up of keys and their values
    - In a list, it is as if the key is the index and the object is the value
    - In a dictionary, instead of the key being the index, the key is another object
- Declaring a dictionary
    - Use curly brackets to enclose it
    - Use a colon to say which key corresponds to the value
    - Use commas to separate each pair
- A key can be a string or number. A value can be anything, including a list.

```
my_dictionary = {“job”:”scientist”, “wage”:”high”,
```

```
“field”:”chemistry”}

#job is the key corresponding to the value scientist
#wage is the key corresponding to the value high
#field is the key corresponding to the value chemistry
```

Challenge Problems

Question

- Create a dictionary called my_dict
- Make one key “friend” and make a list of different names as the value

Answer

```
my_dict = {“friend”:[“Sam”, “Sally”, “Tim”]}
```

Question

- Create a dictionary called profile_dict
- Include the profile of any fictional character
- The keys should include name, age, tv show/book they come from, one friend and anything else you want to include. The values are the traits of that character
- Print the dictionary

Answer

```
profile_dict = {“Name”: “Percy Jackson”, “age”:”16”,
“book”:”The Lightning Thief”, “friend”: “Grover“,
type”:”demigod”}
```

```
print(profile_dict)

#results may vary
```

***** MAKE SURE THEY DON'T DELETE THIS DICTIONARY *****

Teaching - Accessing Values

- Given the key, you can access the value

```
my_dictionary = {"job":"scientist", "wage":"high",
"field":"chemistry"}

my_dictionary["wage"]
#will return "high"

my_dictionary.get("job")
#will return "scientist"
```

Challenge Problems

Question

- Get the name and age of the character from the dictionary they already made using the two different methods and print

Answer

```
profile_dict["Name"]
profile_dict.get("age")
```

Teaching - Reassigning values

- Can change the value given to a key

```
my_dictionary[“wage”] = “medium”
#will change the value assigned to wage to medium
```

Challenge Problems

Question

- Your character grew two years older. Reassign their age value
- Print the new age value

Answer

```
profile_dict[“age”] = “18”
```

Teaching - Adding/Removing keys

- Adding a new key value pair can be done by:

```
my_dictionary[“company”] = “Genentech”
#creates a new key names company corresponding to a value
called Genentech
```

- Removing the key-value pair can be done by:

```
del my_dictionary[“company”]
#deletes company and the value associated with it (Genentech)
```

Challenge Problems

Question

- Your character had a big fight with his or her friend, so they aren’t speaking at the moment. They also got a dog.

- Remove the friend key-value pair, and add dog as a key and its name as the value

Answer

```
del profile_dict[“friend”]

profile_dict[“dog”] = “Snowy”
```

Homework

1: Word Length Dictionary Sorter

Prompt: Using the given dictionary of strings with their corresponding lengths, ask the user to enter a word, place it in the appropriate list, and print out a success message.

```
#PSEUDOCODE

Paste the word dictionary:
word_dictionary = {"3 Letter Words" : ["the", "and", "are",
"for", "not", "but"], "5 Letter Words" : ["sugar", "amber",
"dream", "apple"], "7 Letter Words" : ["Blanket", "England",
"Dancing"]}

Ask the user to input a word

Check if the word is 3, 5, or 7 characters long and place it in
the appropriate list

Print out a success message saying: “The word has been added to
the correct list: [list of words it was placed in]”
```

Else, print: “Sorry, that word cannot be sorted in the dictionary.”

```
#ANSWER

word_dictionary = {"3 Letter Words" : ["the", "and", "are",
"for", "not", "but"], "5 Letter Words" : ["sugar", "amber",
"dream", "apple"], "7 Letter Words" : ["Blanket", "England",
"Dancing"]}
word = input("What is your word? ")

if len(word) == 3:

    word_dictionary["3 Letter Words"].append(word)
    print("The word has been added to the dictionary: " +
str(word_dictionary["3 Letter Words"]))

elif len(word) == 5:

    word_dictionary["5 Letter Words"].append(word)
    print("The word has been added to the dictionary: " +
str(word_dictionary["5 Letter Words"]))

elif len(word) == 7:

    word_dictionary["7 Letter Words"].append(word)
    print("The word has been added to the dictionary: " +
str(word_dictionary["7 Letter Words"]))

else:

    print("Sorry, that word cannot be sorted into the
dictionary.")
```

Python Programming Course 3

Loops

Teaching: While Loops

- What are while loops:
    - A set of statements that are executed as long as a condition is true
        - Will stop once the condition is false

```
>>> i = 1

>>> while i < 6:

   >>> print(i)

   >>> i = i + 1
   # can also be i += 1
```

```
Output: 1, 2, 3, 4, 5
```

- I assign i to be 1
- My while loop tells Python to keep executing this piece of code until i > 6
- I ask Python to print out the variable i
- In my next line of code, I tell Python to assign a new value to i –> i + 1
    - The 2 bullet points above this line of explanation occur together
        - The first time = i is 1
        - Second time = i is 2

- This repeats until i is 5 and once i becomes 6 it won't print the value of i

```
>>> i = 1

>>> while i < 6:

   >>> print(i)

   >>> if i == 3:

      >>> print("You are halfway to the limit")

   >>> elif i == 5:

      >>> print("You have reached the limit")

  >>> i += 1
```

```
Output: 1, 2, 3, You are halfway to the limit, 4, 5, You have reached the limit
```

- Everything is the same as the explanation from above except for a few things:
  - I added that if i equals 3, print "You are halfway to the limit" so Python will execute this particular print statement once the if statement is recognized
  - I added an elif statement that prints "You have reached the limit" when i is equal to 3 so Python will execute this print statement once the elif statement is recognized
  - I moved the "i += 1" to the end so that Python knows not to execute my if or elif statements before printing i
- What is the Break Statement?
  - Allows us to stop the code even if the command is true

        - Useful when you made a mistake in your while loop code and want to change it

```
>>> i = 1

>>> while i < 6:

   >>> print(i)

   >>> if i == 3:

      >>> break

   >>> i += 1
```

Output: 1, 2

- I do the same thing as before: declare a variable i that is equal to 1
- While i < 6 run the code below this statement
    - Print the value of i
    - If i is equal to 3, then break the while loop and come out of it
- The last statement “i += 1” tells to assign i to i + 1 so the loop keeps running without i only being 1

```
>>> i = 1

>>> while i < 6:

>>> i += 1

   >>> if i == 3:

      >>> continue
```

```
>>> print(i)
```

```
Output: 1, 2, 4, 5
```

- I assigned 1 to the variable,i
- The code below the while statement will run until the while statement is false
- Python will add 1 to the variable, i, and assign that new value as i and return to the top of the code
    - If i is 3, then it will recognize the if statement, and continue the code, but continue in this case means to skip over the while loop for that particular value of i
        - This is why the code returns 1, 2, 4, 5, skipping 3
- Challenge Problem

1. Make a variable called i that is equal to any one digit integer
2. Make a while loop that works while i is less than or equal to 10
    a. print(“hello”) the number of times it will take i to reach to 10

Answer:

```
>>> i = 3

>>> while i <= 10:

   >>> num = 10 - i

   >>> print(num)

   >>> i = i + 1
```

Teaching: For Loops

- What are for loops:
    - For loops are loops which iterate over each element in a sequence in Python.
    - As an example, assume we have the list [0, 1, 2, 3, 4]
        - A for loop would iterate over each element (0, 1, 2, 3, and 4), and resultantly, the code in the loop body would be called 5 times (1 for each element.)
        - **If you're still not understanding, that's okay. Here's some code to clarify:**

```
#Note: the range(n) function generates Python lists from [0…n-1]
#    Example: range(3) = [0, 1, 2]

for x in range(8): #declaration of loop

    print("I am called!") #code in loop body
```

```
OUTPUT:

I am called!
I am called!
I am called!
I am called!
I am called!
I am called!
I am called!
I am called!
```

        - As you can see, the loop iterated over the 8 elements in the list, and executed the code in the loop body 8 times.
    - You may have noticed the "x in" component of the loop declaration.

- This is necessary in declaring for loops and extremely useful!
- The x in “x in” tells the Python interpreter to define the local variable x, and as the for loop iterates through each element, x takes its value.
    - Note: The variable does not need to be called x. It can be named anything, as long as it follows the Python variable naming conventions.
- **Here is an example of how this can be useful in code:**

```
#This program will greet all of the guests in a room

guests = [Mark, Steve, Warren, Alex]

for guest in guests:

    print(“Hello “ + guest + “!”)
```

```
OUTPUT:

Hello Mark!
Hello Steve!
Hello Warren!
Hello Alex!
```

- A for loop can utilize the same loop control statements we discussed earlier.
- As you can see, this is an extremely powerful loop.

Challenge Problems

There are 2 challenge problems you will be attempting today.

First Challenge Problem: Create a for loop which executes print(“Hello”) 3 times (Hint: use range())

Second Challenge Problem: Create a for loop which iterates over this list [red, blue, green, purple]. Print out each element, unless the element is green.

Answers:

```
#First Problem

for x in range(3):

    print(“Hello”)

#Second Problem

colors = [“red”, “blue”, “green”, “purple”]

for color in colors:

    if color == “green”:

        continue

    print(color)
```

```
OUTPUT:

#First problem

Hello
Hello
Hello

#Second problem
```

```
red
blue
purple
```

Exceptions

Teaching: Understanding Exceptions

- An error that occurs while code is being executed
- Different types of exceptions include: TypeError, NameError
- Will be shown in the shell

```
>>>6-henry
#name error since henry isn’t a defined variable

>>>6+’7’
#operation isn't possible so it will give you a TypeError
```

Teaching: Handling Exceptions

- try/except is a pair of statements used to handle exceptions
- If no exception; executes whatever is in the try statement
- If there is an exception, executes what is in the except statement
- https://www.programiz.com/python-programming/exceptions has more information about different types of exceptions

```
while x=5:
   try:
     y = int(input(“Give me a number: “)
   except ValueError, TypeError:
     print(“not valid...try again”)
#if the user doesn’t give an integer, it will throw an
exception; the except statement gives the computer an option of
```

```
what to do if a ValueError is thrown

>>> 1 / 0
Traceback (most recent call last):
 File "<string>", line 301, in runcode
 File "<interactive input>", line 1, in <module>
ZeroDivisionError: division by zero
```

Question

- Write a while loop that is always true
- Inside the loop, you should get an int input from the user
    - Then divide 55 by that number
- The except should handle if the number is 0 since that will be undefined

Answer

```
while True:
    try:
      X = int(input(“give me a number: “))
      Y = 55/x
      print(y)
   except ZeroDivisionError:
     print(“don’t input 0”)
```

Teaching: Raising Exceptions

- Raise statement forces an error

```
if int == 0:
   raise ZeroDivisionError(“error error”)
#will raise a NameError
```

Teaching: Clean up Actions

- A statement that is always executed
- Use the keyword finally

```
while True:
    try:
      X = int(input(“give me a number: “))
      Y = 55/x
      print(y)
   except ZeroDivisionError:
     print(“don’t input 0”)
   finally:
     print(“thank you”)
```

Question

- Use the finally statement to print something and see the order it appears in

Answer

- Pictured above

Homework

1: Factorial Calculator

Prompt: Ask the user to enter an integer,and calculate the factorial of the integer. For our purposes, an error message will be printed if a negative number is entered.

```
#PSEUDOCODE

Ask the user to input an integer

Calculate the factorial of the integer that the user enters
(factorial of 4 is 4*3*2*1 = 24)

The factorial of 0 is always 1

For our purposes, let's say that the factorial of a negative
number cannot be calculated

If 0 or a positive integer is entered: "The factorial of [the
integer] is [the value of the factorial]."

Else, print: "The factorial value of a negative number cannot
be calculated."
```

```
#ANSWER

number = input("Enter number: ")
factorial = 1

num = int(number)

if num == 0:
    factorial = 1
```

```
else:
    while (num >= 1):
        factorial *= num
        num -= 1

print(“The factorial of “ + number + “ is “ + str(factorial))
```

Python Programming Course 5

- Functions
    - What is a function and why you need it
    - How do you define a function
    - Returning output from a function
    - Calling and using functions
- Import
    - Libraries and stuff
    - Emoji module
        - Emoji = emojis.encode(“There/’s a :snake: in my boot!”)
        - print(Emoji)
- Object Oriented Programming
    - Overview of object oriented programming
    - How to define classes
        - Attributes
        - Constructors
        - Methods (if time permits)

Functions

Teaching: What is a function?

- Block of code that runs, specifically when it is called

- Good way of organization and reduces the amount of code you need to write (ex: if you have to run a certain for loop in three different places with different sets of data, instead of rewriting it each time, just call a function)

Teaching: Creating a function and calling

- Use define keyword and name the function
- Make sure to include parentheses
- Call it by stating the function name

```
def my_function():
   print(“Hello, how are you?”)
my_function()
```

Challenge Problems

Question

- **Create a function that prints the result of 5 plus 4**

```
def my_function():
   print(5+4)
my_function()
```

Teaching: Arguments

- Information that is passed into a function
- Used if you need to work on data from another part of the code
- When CREATING the function, the argument is a stand in for the actual data that you will input when CALLING the function

```
def my_function(x, y):
   print(x+y)
```

```
my_function(4, 5)
```

Challenge Problems

Question

- **Create a function that and pass a list of strings in**
- **Add “apples” to the end of that list and print it**

```
def my_function(groceries):
   groceries.append(“apples”)
   print(groceries)

grocery_list = [“orange”, “banana”, “bread”]
my_function(grocery_list)
```

Teaching: Return

- Sometimes, the function performs operations, and you need to save the data you get so that it can be used later
- Use the return keyword in the function so that the data can be accessed outside of the keyword
- The result of the function is usually stored in a variable

```
def my_function(x, y):
   z = x + y
   return z
print(my_function(4, 5))
```

Challenge Problems

Question

- **Create a function and pass a string into it**
- **Add each character of the string to a list**
- **Return the list and outside of the function, print the result**

```
def my_function(string_to_convert):
  hello_list = []
  hello_list = list(string_to_convert)
  return hello_list

x = my_function("hello")
print(x)
```

Import

- What are some built-in modules called the “standard library”?
    - Math Modules
        - ceil – returns the smallest integer greater than the number asked for

```
>>> import math

>>> x = 33.7

>>> y = math.ceil(x)

>>> print(y)
```

Output: 34

- I imported the math module because the ceil function was apart of the math module
- I assigned the float 33.7 to the variable, x
- I assigned another variable, y, the value of “math.ceil(x)”
    - It would take 33.7 and find the ceil of it
- Then, I told python to print the value of y
    - What is the smallest integer greater than 33.7?

- 34 – remember rounding up

- floor – returns the largest integer smaller than the number asked for

```
>>> import math

>>> x = 33.99

>>> y = math.floor(x)

>>> print(y)
```

```
Output: 33
```

- I imported the math module because the floor function was apart of the math module
- I assigned the float 33.99 to the variable, x
- I assigned another variable, y, to the value of "math.floor(x)"
  - It would take 33.99 and find the floor of it
- Then, I told python to print the value of y
  - What is the largest integer less than 33.99?
    - 33 – remember rounding down

- range – allows you to find the range of a series of numbers: in the form of a list

```
>>> print(list(range(10)))

>>> print(range(5))
>>> for i in range(10):

   >>> print("This is ", i)
```

```
Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
(0, 5)

This is 0
This is 1
This is 2
This is 3
This is 4
This is 5
This is 6
This is 7
This is 8
This is 9
```

- I printed a list of all of the numbers ranging until 10
    - Default starting is from 0
    - Prints the numbers in the format of a list
- I printed the range of numbers until 10
    - Smallest number: 0
    - Largest number: 9 (because it starts from 0)
- I used a for loop to print out “This is (the value of the particular number at the time)” however many times the range of 10 consisted of

- What are modules?: Python program that contains a related group of functions that can be embedded in your programs
    - A file consisting of Python code
        - You can call this file in another file
        - Allows you to logically organize your code
        - Can define functions, classes, and variables
- How to import modules? (This is how you will mainly use this concept)
    - Import the module you want to use using the “import” statement
        - “Import” statement consists of the following:
            - “import” keyword
            - Name of the module you want to import

- Can be more than one, but need to split all of them by commas

```
>>> import random

>>> for i in range(5):

   >>> print(random.randint(1, 10)
```

```
Output:
3
5
10
1
9
#answers may vary
```

- I imported the random module
  - Allows me to randomly generate a set of values from a given set
- I used a for loop to “print(random.randint(1, 10)” 5 times
- “print(random.randint(1,10)” tells python to print random integers from the range of 1-10
  - 10 <= number >= 1
  - “randint” is a built in function

```
# This is in a different file titled “Greeting”

>>> def greeting(name):

   >>> print(“Hello, ” + name)
```

```
# This is a different file titled “Greeting2”

>>> import Greeting

>>> Greeting.greeting(“Jonathan”)
```

```
Output: Hello, Jonathan
```

```
# This is a different file titled “module1”

>>> def function(words)

   >>> if len(words) % 2 == 0:

      >>> print(“Even”)

   >>> elif len(words) % 2 != 0:

      >>> print(“Odd”)

   >>> else:

      >>> print(“Error”)
```

```
# This is a different file titled “newfile”

>>> import module1

>>> module1.function(“Warrior”)
```

```
Output: Odd
```

- I defined a function called “function” that takes an argument
- If the length of the word entered was even, I told python to return “Even”
- If the length of the word entered was odd, I told python to return “Odd”
- If the length of the word was none of the above, then I told python to return “Error”
- I imported this module1 into another file and then defined a function titled “function” that took “Warrior” as an argument

Challenge Problem:

1. **Make a function that takes two arguments that are integers and returns their sum and product**
2. **If the entered numbers are not integers print “Error”**
3. **Import that file which had the sum and product finder to your new file and assign an argument to the function to test it out**

Solution:

```
>>> def function(int(num1), int(num2)):

   >>> if type(int(num1)) and type(int(num2)) == int:

      >>> print(int(num1) + int(num2))

      >>> print(int(num1) * int(num2))

   >>> else:

      >>> print(“Error”)
```

Object Oriented Programming

Teaching: What Is Object Oriented Programming?

- Currently, we’ve been using a form of programming called procedural programming.
    - In procedural programming, code is executed like a recipe, where a sequence of steps are given and followed by the computer.
- Today, we’ll be starting object oriented programming

    - Object oriented programming gives you a way to structure programs so that properties and behaviors are bundled into individual objects
    - This makes really nice and readable code, which is also very flexible and powerful
- An object could represent a person with a name, property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending
    - As you can see, OOP programs are especially helpful for modeling real world entities in software
- Remember: Objects are the center of object oriented programming (as given by the name) and they not only store as entities to hold data, but also in the overall structure.

Teaching: What Are Classes?

- Focusing first on the data, each thing or object is an instance of some class.
- The primitive data structures available in Python, like numbers, strings, and lists are designed to represent simple things like the cost of something, the name of a poem, and your favorite colors, respectively.
- What if you wanted to represent something much more complicated?
- For example, let's say you wanted to track a number of different animals. If you used a list, the first element could be the animal's name while the second element could represent its age.
- How would you know which element is supposed to be which? What if you had 100 different animals? Are you certain each animal has both a name and an age, and so forth? What if you wanted to add other properties to these animals? This lacks organization, and it's the exact need for classes.

- Classes are used to create new user-defined data structures that contain arbitrary information about something. In the case of an animal, we could create an Animal() class to track properties about the Animal like the name and age.
- It's important to note that a class just provides structure—it's a blueprint for how something should be defined, but it doesn't actually provide any real content itself. The Animal() class may specify that the name and age are necessary for defining an animal, but it will not actually state what a specific animal's name or age is.
- A class is similar to an idea of how something should be defined.

Teaching: What Are Instances?

- While the class is the blueprint, an instance is a copy of the class with actual values, literally an object belonging to a specific class. It's not an idea anymore; it's an actual animal, like a dog named Roger who's eight years old.
- Put another way, a class is like a form or questionnaire. It defines the needed information. After you fill out the form, your specific copy is an instance of the class; it contains actual information relevant to you.
- You can fill out multiple copies to create many different instances, but without the form as a guide, you would be lost, not knowing what information is required. Thus, before you can create individual instances of an object, we must first specify what is needed by defining a class.

- Object Oriented programming is a form of programming that allows users to create their own objects that have methods and attributes

    - Methods are like .append for a list
    - Used to organize and create code that is repeatable especially for larger pieces of code where functions are not enough
- An object is basically an abstract data type created by a developer
    - It can include multiple properties and methods and may even contain other objects.
    - Objects provide a structured approach to programming.

```
>>> Class Dog():
I defined my class and called it dog.
   >>> species = “mammal”
THis a is class object attribute that is the same for any
instance of this class
   >>> def __init__(self, breed, name, spots):
I defined my object and this __init__ followed by the self
keyword indicates that I will be using that as an attribute to
my class or something that I can directly call from my class.
The __init__ indicates an instance of my class and self
indicates an instance of my object.
      >>> self.mybreed = breed
      >>> self.myname = name
      >>> self.myspots = spots
We used the self.breed, and created an attribute or
characteristic of the class, and set it equal to our argument,
breed. You choose the attribute name; it doesn’t have to be
self.breed but it can also be self.dogtype.
   >>> def bark(self,number):
      >>> print(“Woof! My name is {} and the number is
{}.”.format(self.myname,number)
This is a method and will be in open and close parentheses when
called. The self indicates that it uses a particular instance
of my object from my class and that’s why I must say
self.myname in the format(). Since it uses an instance of my
attribute, I must specify self.myname. The number is something
provided by the user so it does not need to be used with
```

self.number.

```
>>> mydog = Dog(breed = “Huskie”, name = “David”, spots = True)
I’m specifying what breed my dog is.
>>> mydog.mybreed
   >>> This will print Huskie
>>> mydog.myspots
   >>> This will print True
>>> mydog.myname
   >>> This will print David
>>> mydog.species
   >>> This will print out mammal
>>> mydog.bark(10)
   >>> This will print out Woof! My name is David and the
number is 10.
I’m calling my breed as an attribute and therefore, my
attribute name has to be what is called. Breed is just my
parameter name, not my actual attribute name so I can’t call
that.
```

```
class Circle():
 pi = 3.14
 def __init__(self, radius = 1):
   self.radius = radius
 def get_circumference(self):
   print(self.radius * self.pi * 2)
 def get_area(self):
   print(self.radius**2*self.pi)


circle = Circle(radius = 30)
circle.get_circumference()
circle.get_area()
```

Homework

1: Creating Various Functions

Prompt: Ask the user to enter 3 integers and perform various functions with these 3 integers as shown in the pseudocode.

```
#PSEUDOCODE

In the main.py, ask a user to enter 3 integers

Create a new file in repl.it and name it VariousFunctions.py

In VariousFunctions.py, make the following functions that each
have 3 integer parameters/arguments:
  1. The name of the function should be maxNum and the function
     should return the greatest of the 3 integers
  2. The name of the function should be minNum and the function
     should return the least of the 3 integers
  3. The name of the function should be averageNums and the
     function should return the average/mean of the 3 integers
  4. The name of the function should be allTheSame and the
     function should return True if all 3 integers are the same
     and False otherwise (if the 3 integers are 4, 4, and 4,
     the function would return True, but if the 3 integers are
     4, 4, and 5, the function would return False)
  5. The name of the function should be allDifferent and the
     function should return True if any 2 of the 3 integers are
     different and False otherwise (if the 3 integers are 4, 4,
     and 4, the function would return False, but if the 3
     integers are 4, 4, and 5, the function would return True)
```

In the main.py, import the VariousFunctions module. Call each function that you created in VariousFunctions.py once and the parameters of the function that are called should be the data collected from user input.

For instance,calling the maxNum function would look like the following:
print(VariousFunctions.maxNum(num1, num2, num3).
Num1, num2, and num3 represent the 3 values collected from user input.

Answer

```
#main.py
import VariousFunctions


num1 = int(input("Enter a number: "))
num2 = int(input("Enter a number: "))
num3 = int(input("Enter a number: "))


print(VariousFunctions.maxNum(num1,num2,num3))
print(VariousFunctions.minNum(num1,num2,num3))
print(VariousFunctions.averageNums(num1,num2,num3))
print(VariousFunctions.allTheSame(num1,num2,num3))
print(VariousFunctions.allDifferent(num1,num2,num3))
```

```
#VariousFunctions.py
def maxNum(num1, num2, num3):
 maxNum = num1
 if(maxNum < num2):
   maxNum = num2
```

```
 if(maxNum < num3):
   maxNum = num3
 return maxNum

def minNum(num1, num2, num3):
 minNum = num1
 if(minNum > num2):
   minNum = num2
 if(minNum > num3):
   minNum = num3
 return minNum

def averageNums(num1, num2, num3):
 return (num1 + num2 + num3) / 3

def allTheSame(num1, num2, num3):
 if(num1==num2 and num2==num3 and num1==num3):
   return True
 else:
   return False

def allDifferent(num1, num2, num3):
 if(num1==num2 or num2==num3 or num1==num3):
   return False
 else:
   return True
```