Week: 02 Date: 02/11/2021

|  |
| --- |
| 15-110 Recitation Week 2 |

# **Reminders**

* HW 1 due Monday 2/15 at Noon EDT!
* Check 1 grades are out! The resubmission deadline is 2/23
* Make sure to watch Prof Kelly’s videos on the Python Translation Process and Python Errors!! The message with the video links is pinned on Piazza
* For recitation today:
	+ There is a starter code file on the website for the Functions Practice Question – use that to get some practice running code!
	+ Recitation feedback form: <https://forms.gle/WKrrbawKktmRu1xp9>

# **Overview**

* Functions, arguments, returned value, side effects
* Basic graphics
* Tokenizing, parsing, translating
* Error types
* Self study: Decimal & binary

|  |
| --- |
| Problems |

# **FUNCTION PRACTICE**

**Before jumping into the function code writing questions, review general function terminology and how function definition and calls work with your students**

def sample(x, y, z):

 answer = x + y + z

print(“This is a side effect”)

 return x + y + z

result = sample(2, 3, 4)

Function Definition:

* def is how you specify to Python that you are writing a function
* sample is the name of the function you are writing (will be used in function calls)
* x, y, z are **parameters** - these are variables with no initial value and are provided values when a function call provides input

Function Body:

* this is where we write our algorithm, which uses the parameters to compute things

Observable Side Effect:

* this is an example of an observable side effect, something that is displayed to the user but is not and cannot be saved for later use

Return Statement:

* we use return statements to output info once the function is done running
* if no return statement is specified, the function by default will return None
	+ remember that None is different from nothing, None is a value in Python

Function Call:

* this is how the user can call the function on specific values, called **arguments** (2, 3, 4), in order to compute something and obtain a result
	+ the value returned by the function will be substituted for the function call when the function is done running
	+ ex: if I write *result = sample(2, 3, 4)*, once sample has finished running on the provided arguments and returns 2 + 3 + 4 = 9, 9 will be substituted for the function call and the variable result will now equal 9
* the number of arguments specified in a function call must match the number of inputs the function expects

**Scope:**

**For the following function, label each variable by its scope.**

time = “8:00 AM”

def alarm(name):

 sound = “BEEP BEEP!!”

 print(sound, “It’s”, time, “ so get out of bed”, name, “!”)

Answer

THE FOLLOWING QUESTIONS ARE AVAILABLE IN THE .PY STATER FILE FOR TODAY!

**Function Parts:**

Write a function *average* that takes 4 numbers as arguments and returns the average of the four numbers. Then, call this function on 4 numbers of your choice and save the result in a variable. **Run through this question with them first and point out the different parts such as definition, input, return value, how to call, etc.**

code

**Type and Builtin Functions:**

Write a function *converter* that takes in an int and a string of a number, converts the string to an int and adds it to the other int, then converts the sum int to a string, and finally returns the string

code

**Print vs. Return:**

Write a function *introduction* that takes in a name and age and prints the message “My name is <name> and I am <age> years old”

code

**Libraries:**

Write a function *area* that uses the *tkinter, math* and *random* modules, takes in a canvas of size 400 by 400, draws a circle of random radius between 0 and 200 in the center of the canvas, and returns the area of the circle.

Code

# **ERRORS**

**Brief intro into how Python works:**

Recall that Python is abstracted - it’s a high level language written for humans to understand. For your computer to be able to understand your code, your interpreter must translate Python code into a language that the computer understands (this language is called bytecode). This translation occurs in three steps:

1. Tokenizing: splitting code into chucks called tokens (based on natural breakpoints in the language grammar)
2. Parsing: grouping tokens
3. Translating: converting the parsed code into bytecode

Errors in Python can occur due to issues with these processes:

If an error occurs during tokenizing or parsing - SYNTAX

If an error occurs after translation while running the bytecode - RUNTIME

There are 3 general categories of errors in Python:

1. Syntax - syntax errors occur when you have not followed the general rules of Python code, these errors occur before your code has even started running
	1. Examples: incorrect indentation, missing parentheses/colons/quotes, incomplete code
2. Runtime - runtime errors occur when your code is being executed and the interpreter is unable to perform a given action
	1. Examples: trying to use an undefined variable name, trying to perform an illegal operation between data types (concatenating a string and an integer), and dividing by zero
3. Logical - logical errors occur when your code is able to run successfully but produces an unintended/incorrect result
	1. These are harder to catch because the interpreter will not be able to tell you what part of your code is incorrect. This is why debugging and testing your code on different inputs is so important!

**For each of the following, identify the type of error and its category:**

1/0

Answer

x = “Hello”

y = “Danielle”

print(x y)

Answer

'2' + 2

Answer

greeting = "hello world)

Answer

print (int( 7.5 ), “Eleven”)

Answer

name = “Elyana’

Answer

a = 3 + 5 7

Answer

def average(num1, num2, num3):

return num1 + num2 + num3 / 3

Answer