Function and Scope

Dr. Dongchul Kim

Function

Functions

- A function is a segment of code that is executed only when it is invoked or called upon. Within a function, you have the option to receive input data, referred to as parameters or arguments.
  - **Input** is not mandatory; you can create a function without any parameters.
  - **Parameters** are the variables declared within the function's definition.
  - **Arguments**, on the other hand, represent the specific values assigned to these variables when the function is called.
- Additionally, a function has the capability to produce and return data as an output.
  - **Output** is not obligatory; you can define a function without any return values.

Defining functions with return

```
def function_name(parameter):    # function head
    statements                    # function body
    return expression             # return statement
```

```
def function_name():              # function head
    statements                    # function body
    return expression             # return statement
```

Defining functions without return

```
def function_name(parameter):    # function head

    statements                    # function body
```

```
def function_name():              # function head

    statements                    # function body
```

Calling a function

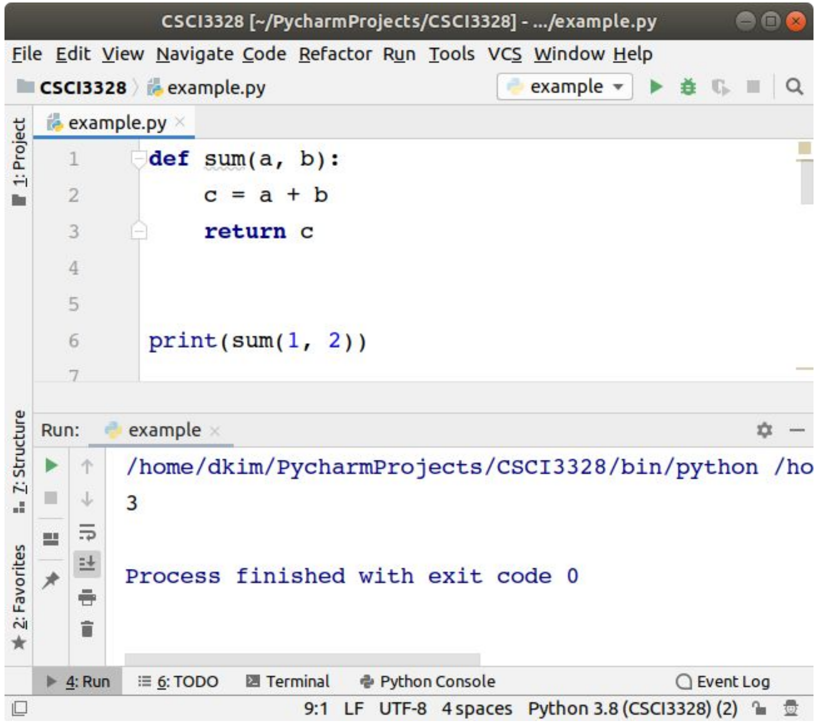
Defining a function does not execute it. Defining it simply names the function and specifies what to do when the function is called.

Calling the function actually performs the specified actions with the indicated parameters. For example, if you define the function `fun` with an integer parameter, you could call it as follows:

```
fun(7)
```

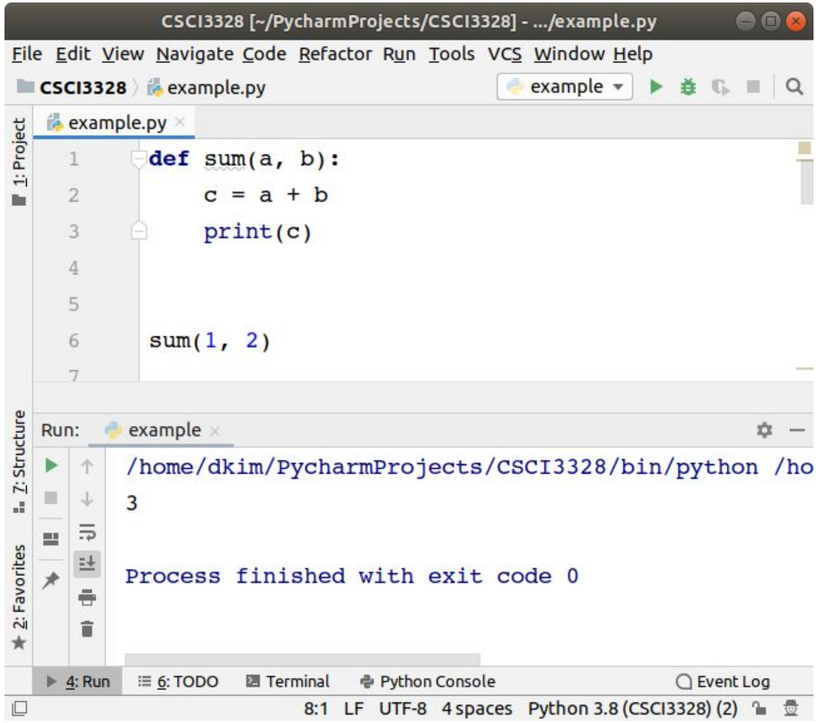
```
fun()
```

Examples


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
Run: example
4: Run 6: TODO Terminal Python Console Event Log
9:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
def sum(a, b):
    c = a + b
    return c


print(sum(1, 2))
```

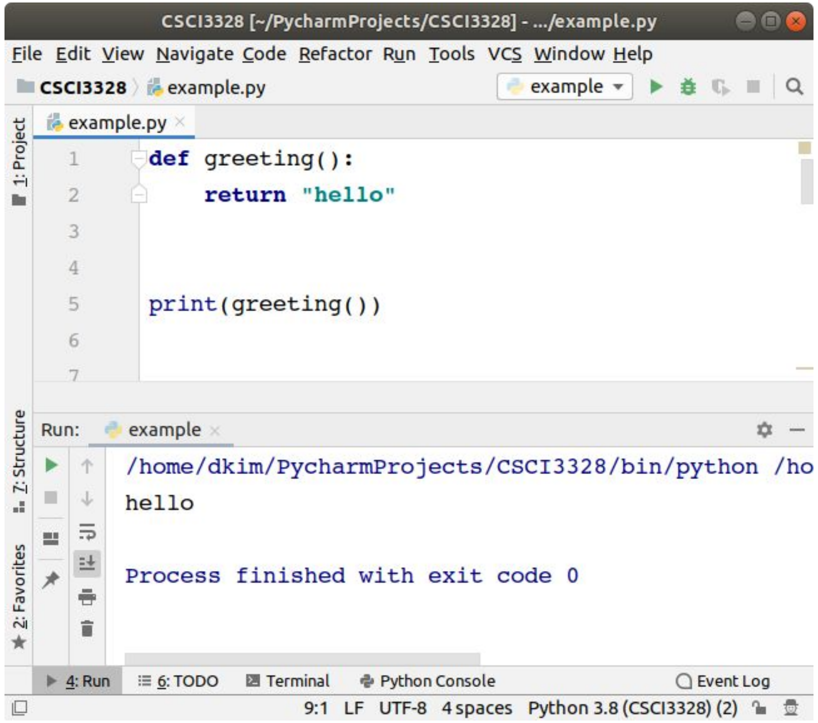
```
/home/dkim/PycharmProjects/CSCI3328/bin/python /ho
3

Process finished with exit code 0
```


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
Run: example
4: Run 6: TODO Terminal Python Console Event Log
8:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
def sum(a, b):
    c = a + b
    print(c)


sum(1, 2)
```

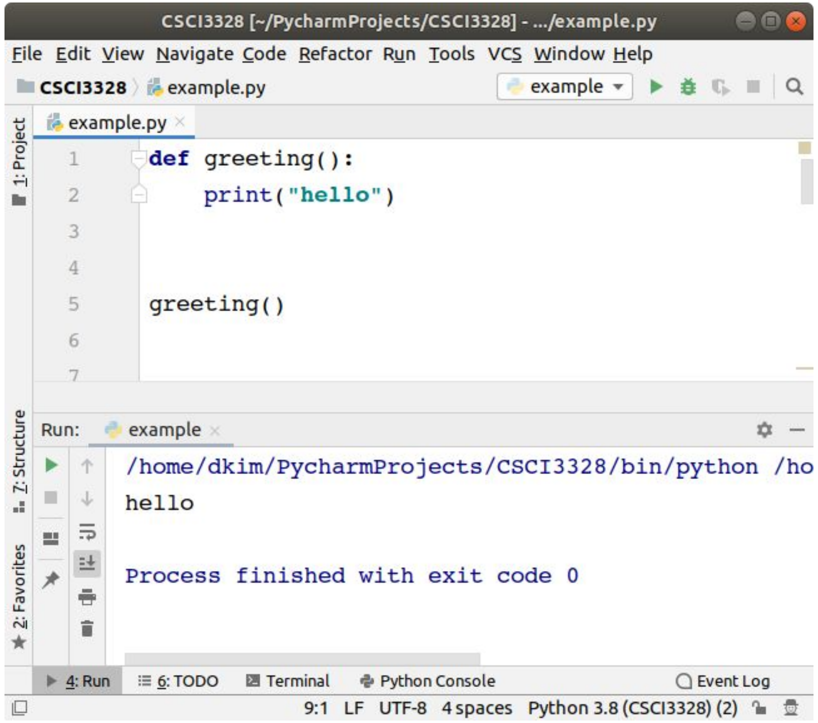
```
/home/dkim/PycharmProjects/CSCI3328/bin/python /ho
3

Process finished with exit code 0
```

Examples


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
Run: example
4: Run 6: TODO Terminal Python Console Event Log
9:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```
def greeting():
    return "hello"


print(greeting())
```

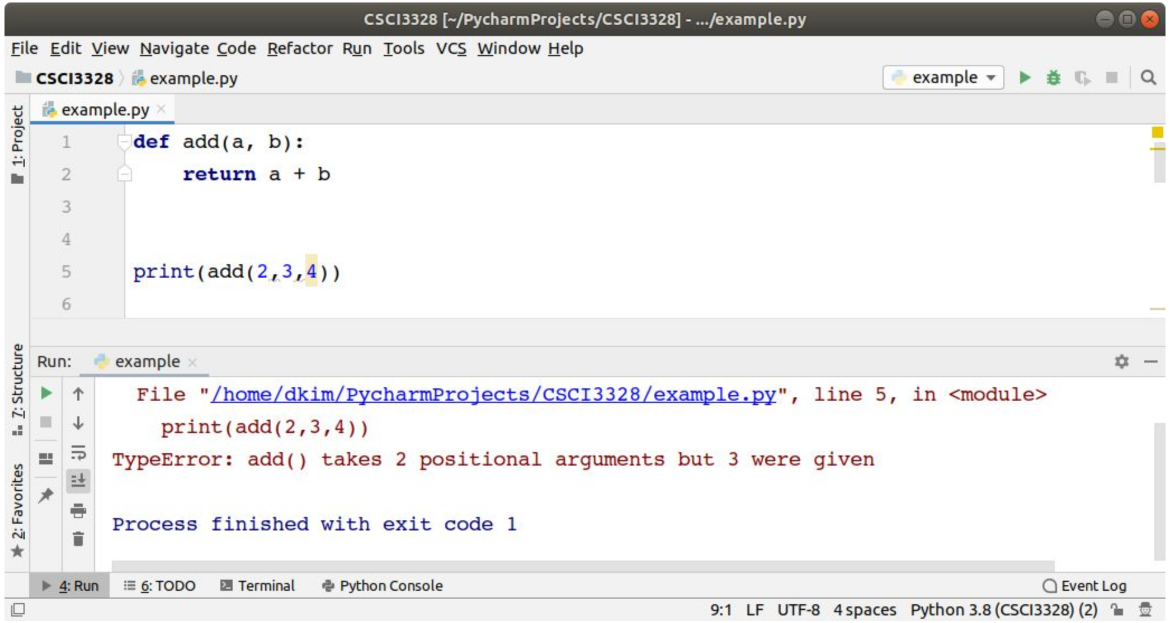
```
/home/dkim/PycharmProjects/CSCI3328/bin/python /ho
hello

Process finished with exit code 0
```


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
Run: example
4: Run 6: TODO Terminal Python Console Event Log
9:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```
def greeting():
    print("hello")


greeting()
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /ho
hello

Process finished with exit code 0
```

Number of parameters


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
Run: example
4: Run 6: TODO Terminal Python Console Event Log
9:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

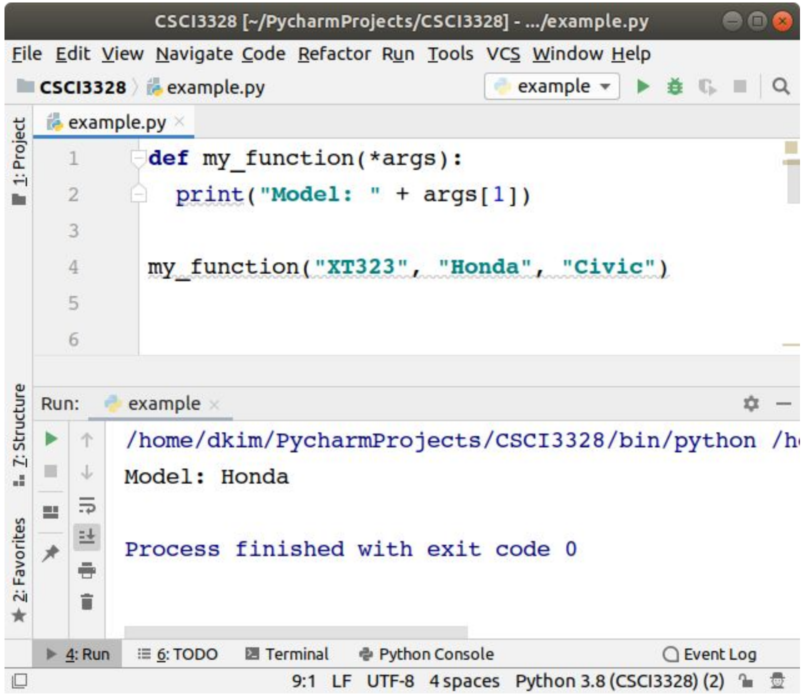
```python
def add(a, b):
    return a + b


print(add(2,3,4))
```

```
  File "/home/dkim/PycharmProjects/CSCI3328/example.py", line 5, in <module>
    print(add(2,3,4))
TypeError: add() takes 2 positional arguments but 3 were given

Process finished with exit code 1
```

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a **tuple** of arguments, and can access the items accordingly:


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
Run: example
4: Run 6: TODO Terminal Python Console Event Log
9:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

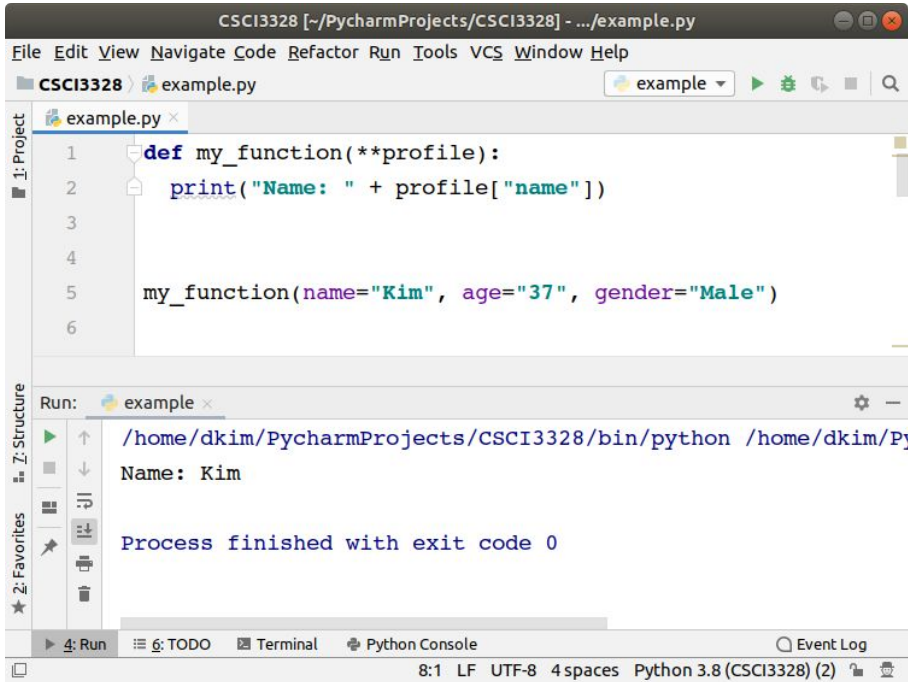
```
def my_function(*args):
  print("Model: " + args[1])

my_function("XT323", "Honda", "Civic")
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /h
Model: Honda

Process finished with exit code 0
```

Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

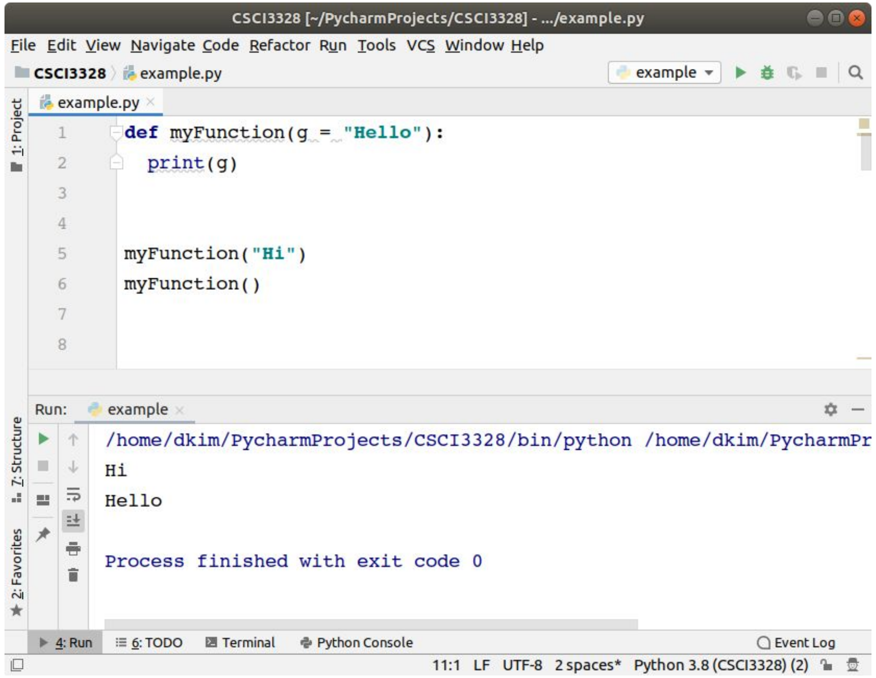

CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328
example.py
example
1: Project
7: Structure
2: Favorites
Run:
4: Run
6: TODO
Terminal
Python Console
Event Log
8:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```
def my_function(**profile):
  print("Name: " + profile["name"])


my_function(name="Kim", age="37", gender="Male")
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Py
Name: Kim

Process finished with exit code 0
```

Default Parameter Value

If we call the function without argument, it uses the default value:


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
Run: example
4: Run 6: TODO Terminal Python Console Event Log
11:1 LF UTF-8 2 spaces* Python 3.8 (CSCI3328) (2)

```python
def myFunction(g = "Hello"):
  print(g)


myFunction("Hi")
myFunction()
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/PycharmPr
Hi
Hello

Process finished with exit code 0
```

Call by value vs Call by reference

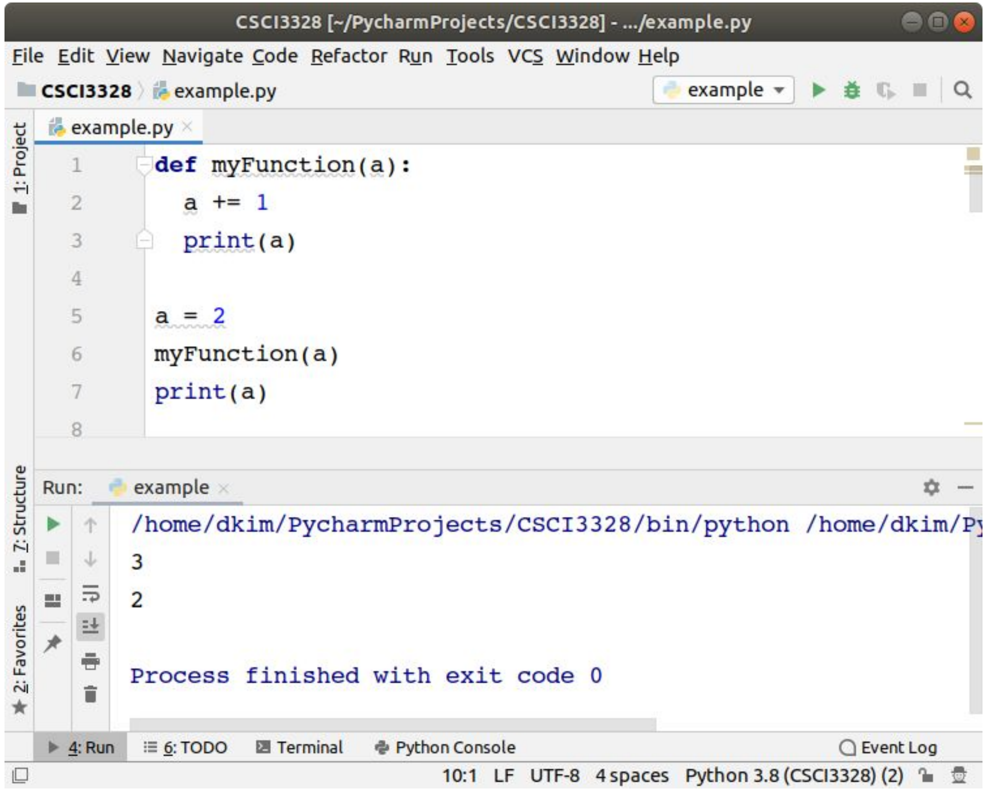
Python utilizes a system, which is known as “Call by Object Reference” or “Call by assignment”.

In the event that you pass arguments like whole numbers, strings or tuples to a function, the passing is like call by value because you can not change the value of the immutable objects being passed to the function.

Whereas passing mutable objects (e.g. list) can be considered as call by reference because when their values are changed inside the function, then it will also be reflected outside the function.

Call by Value vs Call by Reference

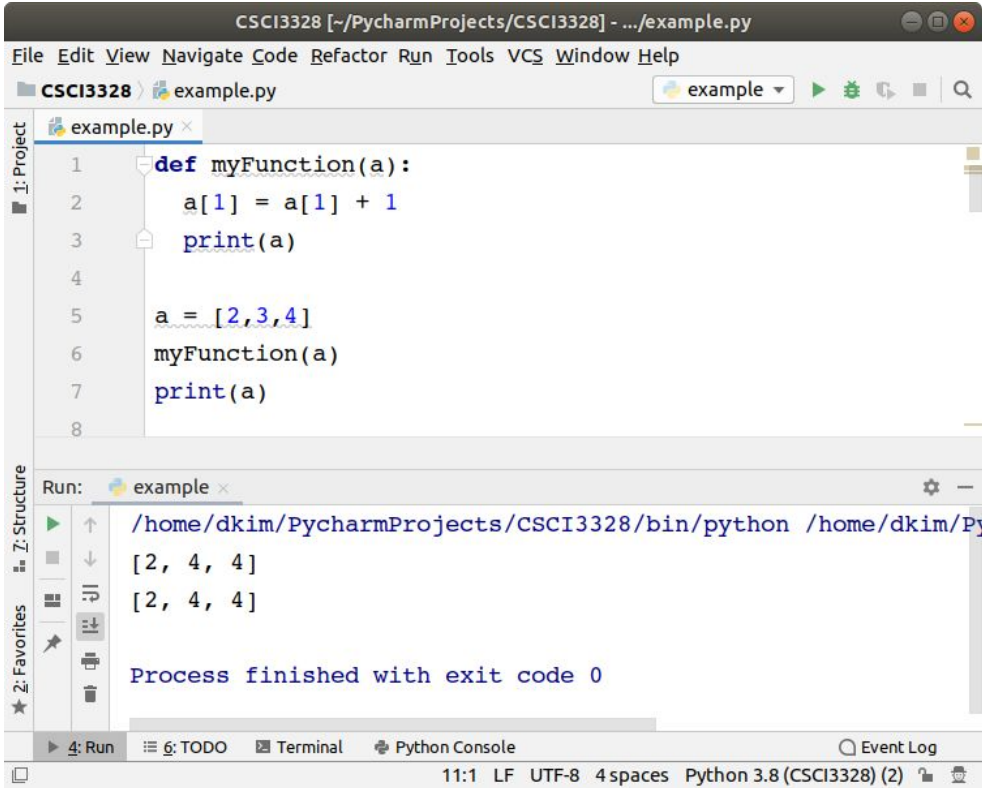
Call-by-value


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run: example
4: Run
6: TODO
Terminal
Python Console
Event Log
10:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
def myFunction(a):
    a += 1
    print(a)

a = 2
myFunction(a)
print(a)
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Py
3
2

Process finished with exit code 0
```

Call-by-reference


CSCI3328 [~/PycharmProjects/CSCI3328] - .../example.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help
CSCI3328 example.py
example
example.py
1: Project
7: Structure
2: Favorites
Run: example
4: Run 6: TODO Terminal Python Console Event Log
11:1 LF UTF-8 4 spaces Python 3.8 (CSCI3328) (2)

```python
def myFunction(a):
    a[1] = a[1] + 1
    print(a)

a = [2,3,4]
myFunction(a)
print(a)
```

```
/home/dkim/PycharmProjects/CSCI3328/bin/python /home/dkim/Py
[2, 4, 4]
[2, 4, 4]

Process finished with exit code 0
```

Scope

Scope

The variable is available only in the region that it is declared. The limitation of such variable declaration is scope. The scope of variable or method defines where those elements are accessible.

```
def func1():
    a = 5
    print(a)

func1()
print(a)
```

The last print statement throws **error** as a is not accessible at the main part of the code.

Scope

One way to have access to the variable at all places is to declare the variable at global scope. Variable created at the main level is global and can be accessed inside the function definition.

```
a = 5
def func1():
    print(a)

func1()
print(a)
```

Now both the print statements have access to the variable.

Scope

When the same name is used for variables inside and outside a function, then python treats them as two separate variables.

```
a = 5
def func1():
    a = 7
    print(a)

func1()
print(a)
```

If we change the value of a inside the function, that change won’t affect the variable outside the function.

Scope

While a variable is defined inside a function, the variable can be declared as global using the “global” keyword.

```
def func1():
    global a
    a = 10
    print(a)

func1()
print(a)
```

Lambda

Lambda function

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression. For example,

A lambda function that adds `10` to the number passed in as an argument, and print the result:

```
x = lambda a : a + 10

print(x(5))
```

Lambda function with multiple arguments

A lambda function that multiplies argument `a` with argument `b` and print the result:

```
x = lambda a, b : a * b

print(x(5, 6))
```

Lab 11-1 ~ 11-4

- Submission
  - Capture the output of your program.
  - Upload both the captured image files and Python files on Blackboard.
  - Create a separate file for each lab; do NOT combine multiple labs into a single file.

Lab 11-1

Define a function that converts US Dollar to Mexico Peso.
The function gets a double value as an argument (dollar)
The function returns a double value (Peso) after a conversion.
After defining the function, call the function and display the result of the conversion. The user inputs a US dollar value using `input()`.
Use this exchange rate (1 Mexican Peso = 0.058 United States Dollar). For example,

```
US Dollar: $100.00 (enter)
Mexico Peso: $ 1715.94
```

Lab 11-1

Function Definition (30 Points):
Accurately defining a function to convert US Dollars to Mexico Pesos (30 Points)
Incorrect or incomplete function definition (0 Points)

Handling User Input (20 Points):
Correctly prompting for and capturing the user's input for the US Dollar amount (20 Points)
Incorrect prompt or failure to capture input (0 Points)

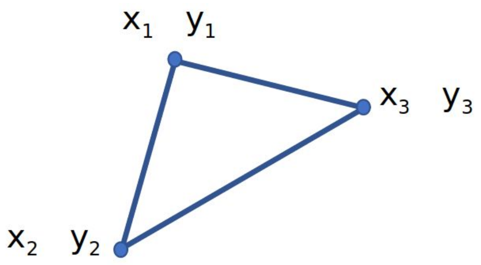
Conversion Logic Implementation (30 Points):
Implementing the correct conversion logic within the function using the current exchange rate (30 Points)
Incorrect or faulty conversion logic (0 Points)

Function Call and Output Display (20 Points):
Correctly calling the function with user input and displaying the result in Mexico Pesos format (20 Points)
Incorrect function call or output display (0 Points)

Total Points: 100

Lab 11-2

Make a function that returns the area of a triangle given three points of the triangle as parameters, and then call the function with test values (6 arguments, x_1, y_1, x_2, y_2, x_3, y_3) that the user inputs like below.



```
Please input three points :  2 1 8 9 1 8 (enter)
The area is 25
```

Lab 11-2

Function Definition for Area Calculation (40 Points):
Correctly defining a function that calculates the area of a triangle given three coordinate points (40 Points)
Incomplete or incorrect function definition (0 Points)

User Input for Coordinate Points (10 Points):
Accurately prompting for and capturing six input values (x1, y1, x2, y2, x3, y3) representing the coordinates of the triangle's points (10 Points)
Incorrect or missing prompt for coordinate points (0 Points)

Correctness of Area Calculation (10 Points):
Implementing the correct formula within the function to calculate the area of the triangle using the provided points (10 Points)
Incorrect area calculation or formula usage (0 Points)

Function Call and Output Display (40 Points):
Successfully calling the function with user-inputted points and displaying the result in the format "The area is [calculated area]" (40 Points)
Incorrect function call or output display (0 Points)

Total Points: 100

Lab 11-3

Define a function that has an integer argument and boolean return value.

The return value indicates whether or not the argument is a prime number.

The function name is prime. Implement the function and call the function with a test value that the user inputs using `input()`.

```
Please input an integer: 13(enter)
13 is a prime number
```

Lab 11-3

Function Definition (prime) (40 Points):
Accurately defining the function prime with an integer argument and a boolean return value (40 Points)
Incorrect or incomplete function definition (0 Points)

Prime Number Logic Implementation (10 Points):
Correctly implementing the logic within the function to determine if the given integer is a prime number (10 Points)
Incorrect or faulty prime number determination logic (0 Points)

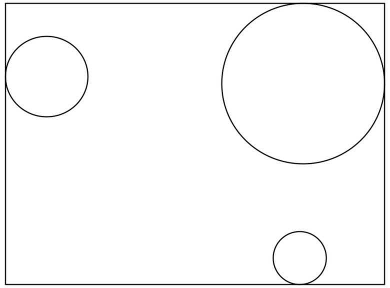
Handling User Input (10 Points):
Correctly prompting for and capturing the user's integer input (10 Points)
Incorrect prompt or failure to capture input (0 Points)

Function Call and Output Display (40 Points):
Successfully calling the prime function with the user input and displaying the appropriate message indicating whether the input is a prime number (40 Points)
Incorrect function call or output display (0 Points)

Total Points: 100

Lab 11-4

Make a program to find the area of a rectangle that covers M circles. The rectangle has only vertical and horizontal edges. The rectangle size should be minimized to cover the circles. The circles can be overlapped. For example, when M = 3,



The input and output format is as follows;

Input

```
M // number of circles
c1x c1y c1r  // origin of 1st circle and its radius
c2x c2y c2r  // origin of 2nd circle and its radius
…
cMx cMy cMr // origin of Mth circle and its radius
```

Output

Area of the rectangle

M is number of circles. Following M lines are for circles. Each line has three values for x, y coordinates (e.g. c1x and c1y) as circle's center and radius (e.g. c1r).
Area in Output is the estimated rectangle area.

Example

Input:

```
3
7.8 3.5 0.5
8.7 8.2 1.2
3.6 7.0 0.8
```

Output:

```
45.44
```

Lab 11-4

Input Capture for M and Circle Details (10 Points):
Accurately prompting for and capturing the number of circles (M) and the details of each circle (center coordinates and radius) (20 Points)
Incorrect or missing input prompt or failure to capture circle details (0 Points)

Rectangle Area Calculation Logic (70 Points):
Correctly implementing logic to calculate the minimum rectangle area that covers the given M circles (40 Points)
Incorrect or faulty rectangle area calculation logic (0 Points)

Correct Output Display (20 Points):
Displaying the calculated rectangle area in the correct format (20 Points)
Incorrect or missing output format (0 Points)

Total Points: 100