Learning Quiz 11: User-Defined Functions

Due Oct 16 at 1pm **Points** 5 **Questions** 5 **Available** until Dec 4 at 11:59pm
Time Limit None **Allowed Attempts** Unlimited

Instructions

Prior to completing this quiz, you should have read:

- Sections 3.3-3.5 (p. 67-81)

Please also go over Practice Problems 3.8 through 3.16 in the textbook (solutions at the end of the chapter) before attempting this quiz.

This quiz was created for learning purposes. You may attempt this quiz as many times as you would like. The highest score *prior to the deadline* will count towards the final course grade. No late submissions will be accepted.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	16 minutes	3.33 out of 5

Score for this attempt: **3.33** out of 5
Submitted Oct 15 at 4:45pm
This attempt took 16 minutes.

Question 1 **0.33 / 1 pts**

In Python, a function can be defined using the **def** keyword. Below, we create a function that *returns* the average of two numbers:

```
def avg1(x,y):
    return (x + y)/2
```

We can also optionally print the average of two numbers:

```
def avg2(x,y):
    print((x + y)/2)
```

Note that while avg1() returns a number, avg2() prints the numbers. Consider avg1() and avg2() functions created and the following two lines of code:

```
var1 = avg1(3, 5)
var2 = avg2(3, 5)
```

Which of the following statements are True? Select all.

- [ ] by creating var1, the avg1() function will print a number
- [ ] var2 is 8
- [ ] var1 is 8
- [x] var1 is 4

Correct!

- [x] by creating var2, the avg2() function will print a number

Correct!

- [x] var2 is 4

ou Answered

- [ ] var1 is None
- [ ] var2 is None

orrect Answer

Question 2

1 / 1 pts

In Question 1, avg1() and avg2() only take two arguments (x and y). You can make it so that a function can optionally different numbers of arguments. In the example below, avg() requires one argument (w) but optionally takes 3 other arguments (x, y, z) which are None by default.

```
def avg(w, x = None, y = None, z = None):
    sum = w
    count = 1
    if x != None:
        count += 1    ## this is the same as count = count + 1
        sum += x      ## this is the same as sum = sum + x
    if y != None:
        count += 1
        sum += y
    if z != None:
        count += 1
        sum += z
    return sum / count
```

Depending on what you want your function to do, your functions do not need to all be the same type.

```
def patientformat(id, last = '', first = '', age = 999):
    print(id, last, first, age)
```

Test different values for the function patientformat(). Which arguments are required? Which are arguments are optional?

Correct! **id** required

Correct! **last** optional

Correct! **first** optional

Correct! **age** optional

Question 3

0 / 1 pts

You can also create functions that take an unlimited number of arguments. The arguments will be treated as tuples. The following function inffun() prints all of the arguments passed through:

```
def inffun(*arg):
    for i in arg:
        print(i)
```

Consider the following function thisfun():

```
def thisfun(x, y, *z):
    ...
```

What is the minimum number of required arguments?

- 3
- orrect Answer 2
- 0
- ou Answered 1
- 4

Question 4

1 / 1 pts

Immutable objects cannot be modified. Mutable objects can be modified. Strings, integers, and tuples are immutable. Lists are mutable. The

following function will work for both strings and lists:

```
def thisfun(thisarg):
    for i in thisarg:
        print(thisarg)
```

Adding this next line will attempt to modify the first element of "thisarg".

```
def thisfun(thisarg):
    for i in thisarg:
        print(thisarg)
    thisarg[0] = 'z'
```

However, you will find that the function will generate an error if "thisarg" is a string while "thisarg" will change the original list even outside of the function. You can use the following lines of code to help test thisfun():

```
thisstr = 'hello'
thisfun(thisstr)
print(thisstr)

thislst = ['hi','hola','hey']
thisfun(thislst)
print(thislst)
```

What is stored in thatlst after running the following lines of code?

```
def modlst(arg1):
    print(arg1)
    arg1[1] = 'garlic'
thatlst = ['onion', 'pickles', 'lettuce', 'tomatoes']
modlst(thatlst)
```

- 'garlic'

Correct!

- ['onion', 'garlic', 'lettuce', 'tomatoes']
- ['garlic', 'pickles', 'lettuce', 'tomatoes']
- ['onion', 'pickles', 'lettuce', 'tomatoes']

Question 5

1 / 1 pts

The following lines of code serve as an example why you'll want to use clear and distinct variable names.

```
def testfun(x):
    print(x)
    x = 3
    print(x)
    return(x)

x = 15
b = testfun(x)
```

It can become difficult to keep track of what x equals in the following lines of code. Note, x in testfun() does not modify the value of x outside of the testfun() function.

What is the value of x and b after running the above lines of code?

- x is 3; b is None
- x is 15; b is None
- x is 15; b is 15
- x is 3; b is 3

Correct!

- x is 15; b is 3

Quiz Score: **3.33** out of 5