Computer Seminar – User-Defined Functions (Week 8)

Contents

Part 1 - Review of Functions (with/without parameters)

Make sure that you understand the changes shown in each of the following examples. Type them in and run them to check your understanding.

a. Function that prints a greeting:

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()  # call function
```

b. Passing information to a function

```
def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username + "!")

user = input('Enter your name ')
greet_user(user)   # call function
```

c. Multiple parameters

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("My " + animal_type + "'s name is " + pet_name + ".")
```

- Multiple function calls can be made to the above function:

```
describe_pet('dog', 'Fido')
describe_pet('cat', 'Felix')
describe_pet('dog', 'Rex')
describe_pet('hamster', 'Bob')
```

d. Times table – define a function

Write a function called *timestable* that will print the 7 times table. Write it so that you can use *timestable(7)* to run it. Check that *timestable(9)* produce the 9 times table. Hints: use range(1, 13) in the function to produce a timetable. E.g., If *timestable(7)* was entered the output would be:

7, 14, 21,28, 35, 42, 49, 56, 63, 70, 77, 84 (either on one line or on separate lines)

Part 2 - Returning data from functions

a. **Grade Classification – using return**
Here is a sample function that could be used to convert a mark to a grade classification. Type it in and use the test data shown below to check it works as expected.

```
#Convert a mark to a grade:
def convertMark(mark):
    if mark >= 70:
        grade = 'First'
    elif mark >= 60:
        grade = '2:1'
    elif mark >= 50:
       grade = '2:2'
    elif mark >= 40:
        grade = 'Third'
    else:
        grade = 'Fail'
    return grade
```

- Use the following test data to call the function and print the returned results.

```
grade1 = convertMark(75)
print(grade1)
grade2 = convertMark(42)
print(grade2)
print(convertMark(55))  #call & print in one statement
```

b. Negative, Positive, Zero – using return
Create a program that contains a function called `negativePositiveZero`. It is passed one numeric (integer or float) parameter. The function should not print anything but return one string values:

- `'negative'` if the number is negative. E.g., less than zero (if);.
- `'positive'` if the number is positive. E.g., greater than zero (elif);
- `'zero'` or if by default the number is zero (else).

The main program code will call the function with the following test values and print the returned results. For each test, the program should print if the numeric is negative, positive or zero.

```
result = negativePositiveZero(-25.7)
print('-25.7 is', result)
result = negativePositiveZero(0.0)
print('0.0 is', result)
result = negativePositiveZero(123.45)
print('123.45 is', result)
```

- For extra practice, allow the user to enter a value. Use that value in your function call and then print the results.

```
# Get user input and call the function.
userValue = input('Enter a number: ')
userValue = float(userValue)
userResult = negativePositiveZero(userValue)
print(userValue, 'is', userResult)
```

Part 3 - Default arguments

a. Remember the program from Part 1c. Multiple parameters:

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("My " + animal_type + "'s name is " + pet_name + ".")

describe_pet('dog', 'Fido')  # Call the function
```

- If the most common animal type is dog, then we can add this as a default argument. Then, if the animal is a dog then the animal_type argument can be omitted from the function call.
- It is necessary to change the order of the parameters in the function definition because a default parameter cannot be followed by a parameter that is not default.

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print("My " + animal_type + "'s name is " + pet_name + ".")

#Test data to call the function
describe_pet('Rover')
describe_pet('Ginger', 'cat')
describe_pet('Bobby', 'dog')
```

b. Write a function that contains a default value

- Write a function that accepts the name of a programming language and prints a statement such as "My favourite programming language is Python".
- The function definition should contain a parameter with a default value, such as Python.
- Call your function at least three times. Make sure at least one of the calls includes an argument, and at least one call does not include an argument.

Part 4 - Using main()

This shows an example of using a main() function. It defines the arguments to pass to the functions, calls the functions, and output the results. We have functions *mysum()* and *myproduct()* that we are running and testing using a *main()* function.

```
def mysum(x, y):
    """
    Takes two input arguments x and y and returns their sum
    """
    s = x + y
    return s

def myproduct(x, y):
    """
    Takes two input arguments x and y and returns their product
    """
    s = x * y
    return s
```

```
def main():
  a = 5
  b = 3
  print(mysum(a, b))                    # hopefully outputs 8
  print(myproduct(a, b))               # hopefully outputs 15

main()
```

Alternative main

- Before the Python interpreter executes a program, it defines a special variable called __name_ _ and sets the value to:
  - **"__main__" when the program is being executed by itself.**
  - Or the module name if the program is being imported by another program.

- **Therefore, we can use an if statement to check if the program is standalone (value is "__main__")** or being used by another program. Then select to execute code based on value. Your programs will be standalone:

```
if __name__ == "__main__":
   main()
```

Part 5 - Extended exercise: Creating a shopping cart

- This exercise will create a program that resembles a shopping cart. The program will store products within a list.
- The program will also use a main function that will contain the loop and handle user input.
- You will create functions to do the tasks of adding, removing, clearing, and showing cart items.

- The program will use a list to hold the products added to the *cart*. This starts off as an empty list.
  - Declare *cart* outside of the functions that you define to make it global and allow us to work with it throughout the program.
  - Using a list will also allow us to edit the variable directly without having to pass it around because of how item assignments work.

```
# list variable (declare outside your user-defined functions to make it global)
cart = [ ]
```

- You may need to include the **pass** statement in empty functions until the code for that function is written.
- Throughout the program use f-string formatting to format output in print statements.

Steps:

1. Create the following functions first:
   - showCart()   (this function is called for tasks **quit/remove/show)**
   - addItem(item)
   - removeItem(item)
   - clearCart()
2. Add the main() function (shown below)
3. Test that the program runs as expected.

Step 1 - Create the following functions first.

Adding items

- Create the function for adding items to the *cart* list. We will call this function later when we create the main loop. When called, append the *item* passed into the parameter, and output to the user.

```
def addItem(item):
    """   Takes item and appends it to cart """
    # append item to cart
    # print item is added
```

Removing Items

- Next, create the function that will remove items from the cart list.
- Removing an item that doesn't exist would cause the program to crash. Therefore, make a check that the product is in the list before attempting to remove it. An alternative approach is to use try/except.

```
def removeItem(item):
    """  Takes item and removes it from the cart """
    # check if the item is in the list
    # if item is in list remove it with the remove operation.  Print item removed
    # otherwise, print item could not be removed
```

Show Items

- Create the function to show the items in the cart. This function is called for tasks **quit/remove/show.**

```
def showCart():
    """  Prints items in cart """
    # check if there are items in the cart.
    # If it's empty, let the user know.
    # Otherwise, loop over the items and output one per line.
```

Clear cart

- Create the function to clear items from the cart. Use the **clear()** method to remove all items from the list.

```
def clearCart():
    """
      Clears items from cart and prints that cart is empty
    """
    # Use clear() method to empty cart
    # Print cart is empty
```

Step Two – Add the main() function

Type in the following main function. This will loop until the user enters 'quit'.

```
def main():
  done = False

  while not done:
        ans = input('quit/add/remove/show/clear: ').lower()
        if ans == 'quit':
            print('Thanks for using our program.')
            showCart()                # write this function
            done = True
        elif ans == 'add':
            item = input('What would you like to add? ').title()
            addItem(item)             # write this function
        elif ans == 'remove':
            showCart()
            item = input('What item would you like to remove? ').title()
            removeItem(item)          # write this function
        elif ans == 'show':
            showCart()
        elif ans == 'clear':
            clearCart()               #write this function
        else:
            print('Sorry that was not an option.')

main()     # run the program
```

Step 3 - Test that the program runs as expected.