Csci 1523
1523 Study Guide Module 05 - Functions

Student (Print): Amethyst O'Connell

This study guide contains 9 pages (including this cover page) and 37 problems. Check to see if any pages are missing. Enter all requested information on the top of this page, and put your initials on the top of every page, in case the pages become separated.

You may use your books, notes, calculator or internet sources while completing this study guide.

Please try to answer the sections clearly and PRINT your answers legibly.

Chapter 5 Dierbach Study Guide
Problems we attempt to solve on a digital computer can be large and very complex. It is often in our interest to break these complex problems into smaller more easily understood problems then consolidate them into a comprehensive solution which we can implement.

This process first *abstracts* the problem at hand and allows us to *decompose* it into smaller more manageable subproblems. When we develop coding implementations that address these subproblems we refer to them as *program routines*.

Once the routines required for the problem at hand are identified and the code for them is developed we assemble them into the program which addresses our complex problem. This is the fundamental way software is developed for all but the smallest programming problems.

Languages such as Python provide syntax which supports such coding efforts called *functions*. *Functions* support the *functional or structured* software development methodology paradigm.

Below find a set of questions concerning the use of functions in the Python language. Use your text or other resources when answering each of these.

1. Program routines are implemented in Python using function.
2. F While functions are a convenient means of implementing program codes they suffer from only being able to be called once.
3. Once completed a function:
   - [✓ A. continues executing the next line of code following the end of the function.]
   - [✓ B. can be sent via program labels to another line within the program.]
   - C. returns to the line preceding the one in which it was called.
   - [✓ D. returns to the line following the one in which it was called.]
   - E. completes program execution and stops.
4. T As in Language C functions must return a unique value, typically this is an integer, but not always.
5. Which of the following can be found in a function header (choose all that apply):
   - A. return type
   - [✓ B. the **def** keyword]
   - [✓ C. an identifier which is the function's name]
   - D. a list of integers
   - [✓ E. a list of formal parameters]
   - F. the function body
   - [✓ G. the **:** symbol]

6. When calling a function formal parameters are passed into the function and actual arguments receive these values.

7. If a function returns a value it must contain a return statement, the value being returned by the function must be on the same line as the return statement.

8. An expression evaluates to a value which are found on the return statement.

9. A non-value-returning function is called not for a returned value but for its side effects.

10. A statement is an action other than returning a function value, such as displaying output on the screen.

11. The code below is extracted from Figure 5-7 in your text. The questions which are shown below are related to this listing.

```
 1 # Temperature Conversion (Celsius-Fahrenheit / Fahrenheit-Celsius)

 3 def displayWelcome():
 4     print('This program will convert a range of temperatures')
 5     print('Enter (F) to convert Fahrenheit to Celsius')
 6     print('Enter (C) to convert Celsius to Fahrenheit\n')

 8 def getConvertTo():
 9     which = input('Enter selection: ')
10     while which != 'F' and which != 'C':
11         which = input('Enter selection: ')
12     return which

14 def displayFahrenToCelsius(start, end):
15      print('\n   Degrees', '  Degrees')
16      print('Fahrenheit', 'Celsius')

18      for temp in range(start, end + 1):
19         converted_temp = (temp - 32) * 5/9
20         print('    ', format(temp, '4.1f'), '     ', \
21             format(converted_temp, '4.1f'))

23 def displayCelsiusToFahren(start, end):
24      print('\n   Degrees', '  Degrees')
25      print('   Celsius', 'Fahrenheit')

27      for temp in range(start, end + 1):
28         converted_temp = (9/5 * temp) + 32
29         print('    ', format(temp, '4.1f'), '     ', \
30             format(converted_temp, '4.1f'))

32 # ---- main

34 # Display program welcome
35 displayWelcome()

37 # Get which conversion from user
38 which = getConvertTo()

40 # Get range of temperatures to convert
41 temp_start = int(input('Enter starting temperature to convert: '))
42 temp_end = int(input('Enter ending temperature to convert: '))
43 # Display range of converted temperatures
44 if which == 'F':
45     displayFahrenToCelsius(temp_start, temp_end)
46 else:
47     displayCelsiusToFahren(temp_start, temp_end)
```

(a) Which of the functions in the listing above return a value:

getConvertTo()

(b) What are the formal parameters in the function *displayFahrenToCelsius*?

start, end

(c) The program above outputs temperatures using a format of 4 characters with one decimal point in the fractional part. On which lines contain this format? In the space below rewrite each of these lines so that the output will display 6 characters with 2 decimal points in the fractional part.

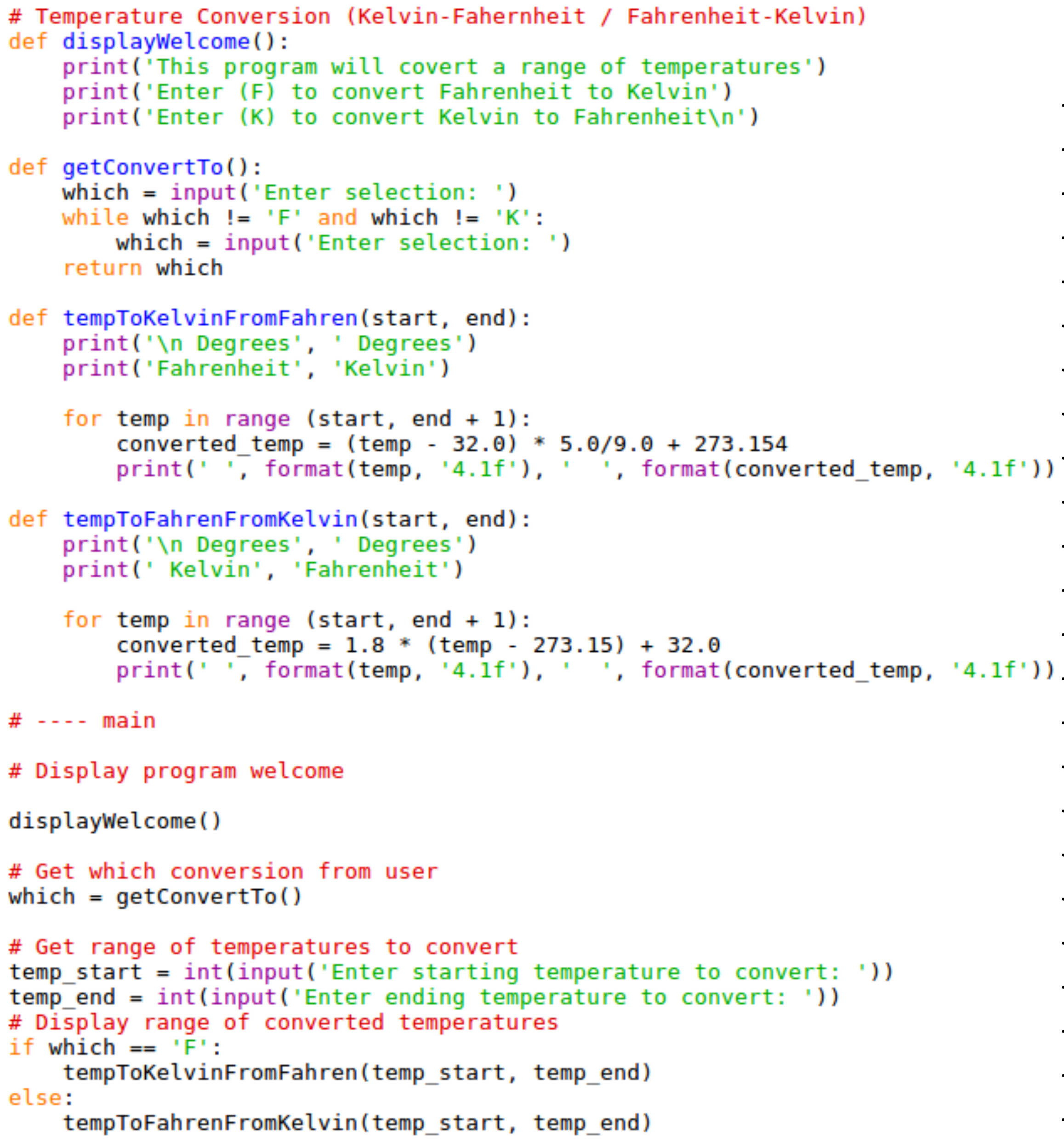
20, 21 and 29, 30

```
print('      ', format(temp, '6.2f'), '   ', format(converted_temp, '6.2f'))
```

(d) What are the identifiers used as arguments in the above listing: start and end.

(e) In the space provided below write out two new functions *tempToFahrenFromKelvin(start, end)* and *tempToKelvinFromFahren(start, end)* using the following formulas for these conversions:

$$Fahrenheit = 1.8x(Kelvin - 273.15) + 32.0$$
$$Kelvin = (Fahrenheit - 32.0) * 5.0/9.0 + 273.154$$



```
# Temperature Conversion (Kelvin-Fahernheit / Fahrenheit-Kelvin)
def displayWelcome():
    print('This program will covert a range of temperatures')
    print('Enter (F) to convert Fahrenheit to Kelvin')
    print('Enter (K) to convert Kelvin to Fahrenheit\n')

def getConvertTo():
    which = input('Enter selection: ')
    while which != 'F' and which != 'K':
        which = input('Enter selection: ')
    return which

def tempToKelvinFromFahren(start, end):
    print('\n Degrees', ' Degrees')
    print('Fahrenheit', 'Kelvin')

    for temp in range (start, end + 1):
        converted_temp = (temp - 32.0) * 5.0/9.0 + 273.154
        print(' ', format(temp, '4.1f'), '   ', format(converted_temp, '4.1f'))

def tempToFahrenFromKelvin(start, end):
    print('\n Degrees', ' Degrees')
    print(' Kelvin', 'Fahrenheit')

    for temp in range (start, end + 1):
        converted_temp = 1.8 * (temp - 273.15) + 32.0
        print(' ', format(temp, '4.1f'), '   ', format(converted_temp, '4.1f'))

# ---- main

# Display program welcome

displayWelcome()

# Get which conversion from user
which = getConvertTo()

# Get range of temperatures to convert
temp_start = int(input('Enter starting temperature to convert: '))
temp_end = int(input('Enter ending temperature to convert: '))
# Display range of converted temperatures
if which == 'F':
    tempToKelvinFromFahren(temp_start, temp_end)
else:
    tempToFahrenFromKelvin(temp_start, temp_end)
```

(f) What changes would need to be made to *getConvertTo()* in order to utilize these new functions in the program. You may code this or simply explain in English.

See above

(g) What changes would need to be made to *displayWelcome()* in order to utilize these new functions in the program. You may code this or simply explain in English.

See previous page

(h) What changes would need to be made to the overall script in order to utilize these new functions in the program. You may code this or simply explain in English.

See previous page

12. Value-returning function calls are:

    A. parameters

    B. arguments

    C. methods

    [✓ D. expressions]

    E. statements

    F. exceptions

13. T Value returning functions cannot have *side effects*.

14. An expression may contain multiple function calls which of the following observations is true about such expressions:

    A. a function call may contain function calls as arguments

    B. conditional expressions may contain function calls

    C. the arguments in a *print* function may contain function calls

    [✓ D. All the above.]

    E. None of the above.

15. T Value-returning functions may return a single value of any type.

16. T Calls to non-value returning functions are *statements*.

17. formal parameters are found within the parentheses of a function call and ________ are found in the function header.

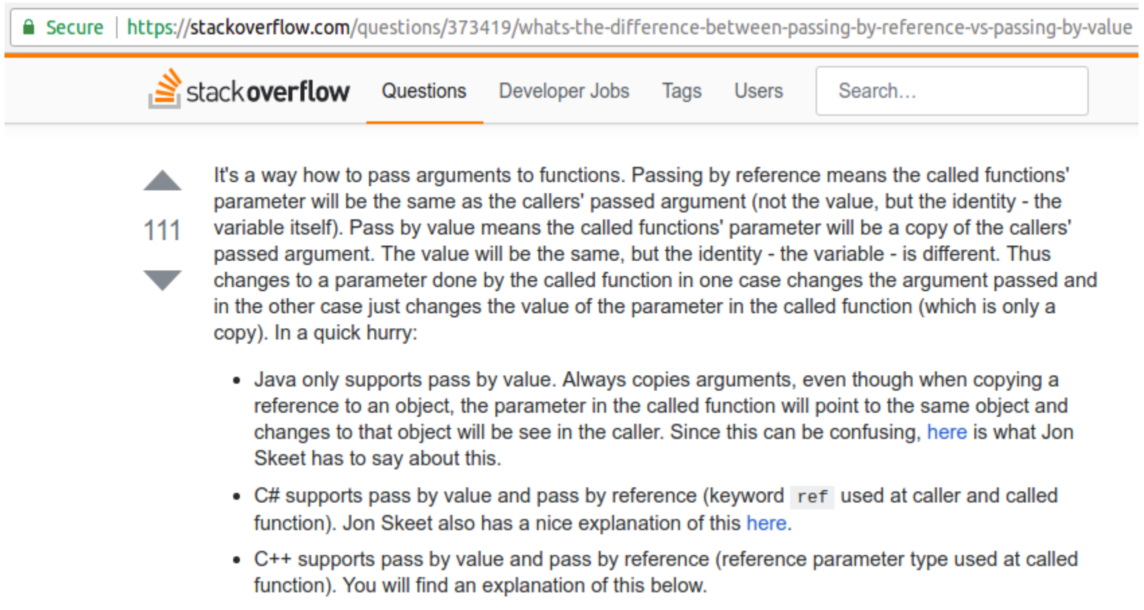
18. F Actual arguments and formal parameters mean the same thing.

19. T The correspondence of actual arguments and their corresponding formal parameters is based on carefully matching argument variable names with parameter variable names.

20. Carefully consider the terms *mutable data type* and *immutable data type*. Explain the difference in the space provided below:

    A mutable data type is changable, an immutable data type is unchangable.

21. The method of calling functions and passing arguments to them in Python uses a technique called *pass-by-value*. Research this and in the space provided below write out a brief explanation of *pass-by-value*.


Secure | https://stackoverflow.com/questions/373419/whats-the-difference-between-passing-by-reference-vs-passing-by-value

It's a way how to pass arguments to functions. Passing by reference means the called functions' parameter will be the same as the callers' passed argument (not the value, but the identity - the variable itself). Pass by value means the called functions' parameter will be a copy of the callers' passed argument. The value will be the same, but the identity - the variable - is different. Thus changes to a parameter done by the called function in one case changes the argument passed and in the other case just changes the value of the parameter in the called function (which is only a copy). In a quick hurry:

- Java only supports pass by value. Always copies arguments, even though when copying a reference to an object, the parameter in the called function will point to the same object and changes to that object will be see in the caller. Since this can be confusing, here is what Jon Skeet has to say about this.
- C# supports pass by value and pass by reference (keyword `ref` used at caller and called function). Jon Skeet also has a nice explanation of this here.
- C++ supports pass by value and pass by reference (reference parameter type used at called function). You will find an explanation of this below.

22. Python supports data types which are *mutable* and those which are *immutable* in the space provided below for each list out these types:
    (a) Immutable types:
    cannot be changed
    (b) Mutable types:
    can be changed

23. Only arguments of a data type that is mutable may be altered within a Python function.

24. F It is generally good programming practice to avoid returning values through altered parameters.

25. Python typically relies upon the position of an argument in a function call to assign it to its corresponding formal parameter. keyword argument override positional arguments.

26. default arguments are values which are assigned to formal parameters in the event the arguments are not provided in the call to the function.

27. T To properly pass an argument using the a keyword parameter one must provide the name of the keyword formal parameter as the variable in an assignment statement as an argument.

28. F All default arguments must precede the positional arguments in a function definition.

29. The code below is contains a function with default and keyword arguments. The function *ie* returns the interest earning of an amount of principle,*(pr)*, invested for a number of periods,*(p)* at a rate of interest,*(i)*. Following it are a series of function call to it in which the arguments are varied. In the spaces provided below write the number which prints to the screen based on the print statement corresponding to that part.

```
def ie(p, i=0.06, pr=1000.0):
    return (1 + i)**p*pr

print(ie(1)) # Part A
print(ie(5, .1)) # Part B
print(ie(5, .1, 2000)) # Part C
print(ie(5, pr=3000, i=0.1)) # Part D
print(ie(pr=3000, i=0.1, 5)) # Part E
print(ie(p=5, pr=3000, i=0.1)) # Part F
```

(a) Part A: 1060.0
(b) Part B: 1610.5100000000004
(c) Part C: 3221.020000000001
(d) Part D: 4831.530000000002
(e) Part E: positional argument follows keyword argument
(f) Part F: 4831.530000000002

30. Local variables are variables which can only be accessed within a function, such variables have local scope.

31. Any variable assigned a value within a function becomes a local variable.

32. Examine the code below in which each output statement is labeled with a Part to be answered below.

33. The time a variable exists is called its lifetime.

34. The lifetime of a local variable begins when it is called in a function and ends when the function is exited.

35. A global variable is defined outside of any function definition. Such variables have global scope.

36. F It is generally considered good programming practice to use global variables over local variables since this makes them available within all functions.

37. F A global variable defined within a Python script is available to functions which appear in that script prior to the global variable being defined.