COMPSCI 101

Principles of Programming

Lecture 15 – Revision Exercises
from Semester 1 Test 1

COMPSCI 101 Test 1

CompSci101 Test 1 Tuesday 1st September at 6pm (NZ Time)

- The test will not be held on campus this semester as we are not back on campus until after the mid-semester break.
- You will be sitting the test using your own computer in your own environment.
- All test questions will be answered and validated using CodeRunner3.
- Date and start time: **6pm (NZ Time) Tuesday 1st September**.
- Worth 20% of your final mark.
- You will have 2.5 hours to complete the test and will finish at 8:30pm (NZ Time) .
- The test has been designed to take a well-prepared student 2 hours but we have added a further 30 minutes to allow for any technical issues.

COMPSCI 101 Test 1

CompSci101 Test 1 Tuesday 1st September at 6pm (NZ Time)

- During the test:
  - you must not discuss the contents of the test with any other person either in person or online.
  - You must not copy code from anyone else or from anywhere on the Internet.
  - you must not share your code or discuss your code with anyone else.
- There are 12 questions and the test is out of 20 marks.
- The questions are in approximate order of difficulty with the easiest questions at the start and the more difficult ones at the end.
- Material covered: Lectures 1 - 12 (inclusive) and Labs 1 - 4 (inclusive).
- It is recommended that you use IDLE to create a Python file to test your code before pasting it into the CodeRunner3 answer box.
- Please remember to submit your test on CodeRunner3 **before 8:30pm (NZ time) on Tuesday 1st September.**

COMPSCI 101 Test 1: Academic Integrity

- By approaching your work with honesty and integrity you actively seek opportunities to learn something new, deepen your understanding, and grow personally.
- The test is unsupervised, so lecturers and markers will take extra time to detect plagiarism, cheating, and copying.
- Cheating means that you are more likely to enter your next course unprepared.
- We ask you to complete the tests in a way that honestly and fairly demonstrates your knowledge and abilities. Take pride in your work :)

We are here to help!

We understand that these are challenging times for all of you. If you need support or help with your studies, or face any obstacles, please get in touch with the COMPSCI 101 teaching team as early as possible. We can often help.

A Note Regarding Test 1 Semester 1, 2020

Some questions in the revision test 1 from semester 1, 2020 include material that won't be tested in this semester's test (see earlier slide for the material that will be covered). However, you have been taught all of this material and it is good practice for your learning.

Question 1

Complete the function **print_greeting(name)** that takes a single parameter, the string **name** and **prints** a greeting using this string. A couple of examples of the function being used are shown below:

```
def print_greeting(name):




def main():
   print_greeting("Damir")
   print_greeting("Ann")

main()
```

```
Hello Damir how are you?
Hello Ann how are you?
```

Question 2

Complete the function **do_multiplication(num1, num2)** that takes two **integer** parameters. The function **returns** the product of these two integers. A few examples of the function being used are shown below.

```
def do_multiplication(num1, num2):



def main():
   print("2 * 3 =",do_multiplication(2,3))
   print("5 * 4 =",do_multiplication(5,4))

main()
```

```
2 * 3 = 6
5 * 4 = 20
```

Question 3

Complete the function **to_the_power_of()** which **prompts** the user to enter two integer values. You can assume that the user will always enter integer values. The function **returns** a string with the answer of taking the first number to the power of the second number. A couple of examples of the function being used are shown below:

```
def to_the_power_of():



def main():
   print(to_the_power_of())
   print(to_the_power_of())

main()
```

```
Number 1: 2
Number 2: 6
2 to the power of 6 is 64

Number 1: 5
Number 2: 2
5 to the power of 2 is 25
```

Question 4

Complete the **get_ordered_words(word1, word2)** function which is passed two strings as parameters. The function **returns** the string which is made up of the longer of the two parameter words followed by a space followed by the shorter of the two parameter words.

Note: you can assume that the two parameter words have different lengths.

```
def get_ordered_words(word1, word2):



def main():
    print(get_ordered_words("out", "dinner"))
    print(get_ordered_words("dinner", "out"))
    print(get_ordered_words("bb", "a"))
main()
```

```
dinner out
dinner out
bb a
```

Question 5

Complete the **get_number(number1, number2)** function which is passed two **positive** integers as parameters. The function returns the **integer** made up of the first digit from the first parameter followed by the last digit of the second parameter.

```
def get_number(number1, number2):



def main():
   print(get_number(76325, 321))
   print(get_number(3, 2))
   print(get_number(50004, 2))
main()
```

```
71
32
52
```

Question 6

Complete the function **print_pattern(number)** which takes a single integer parameter. The pattern **printed** depends on the parameter passed and a few examples of the function being used are shown below. You can assume that the parameter is always a single-digit number.

```
def print_pattern(number):



def main():
   print_pattern(6)
   print_pattern(4)
main()
```

```
666666
 55555
  4444
   333
    22
     1
4444
 333
  22
   1
```

Question 7

Complete the **get_list_of_four_letter_words(words_list)** function which is passed a list of strings as a parameter. The function **returns** a new list of all the words from the parameter list which are four characters in length. All the words in the returned list should be in lowercase characters.

Note 1: You MUST use the *append()* method to add elements to the end of the list.

Note 2: If the parameter list is empty or contains no words which are four characters in length, the function returns an empty list.

```
def get_list_of_four_letter_words(words_list):


def main():
   words = get_list_of_four_letter_words(['into', 'elephant', 'room', 'the'])
   print(words)
   print(get_list_of_four_letter_words(['hole', 'down', 'the', 'goes', 'rabbit']))
main()
```

```
['into', 'room']
['hole', 'down', 'goes']
```

Question 8

Complete the **get_count_containing_9(numbers_list)** function which is passed a list of integers as a parameter. The function **returns** the count of all the numbers in the parameter list which contain the digit 9.

```
def get_count_containing_9(numbers_list):


def main():
    count_contain_9 = get_count_containing_9([393, 6369,
    2042, 40, 9447])
    print(count_contain_9)
    print(get_count_containing_9([191, 45594, 1241, 9,
    929]))
    print(get_count_containing_9([465, 383, 282]))
main()
```

```
3
4
0
```

Question 9

Complete the **contains_mostly_vowel_ending_words(words_list)** function which is passed a list of strings as a parameter. The function **returns** *True* if **ALL** the words in the parameter list which have a length greater than 2 end with a vowel, otherwise the function returns *False*.

Note: if there are no words in the parameter list which have a length greater than 2, the function returns *True*.

```
def contains_mostly_vowel_ending_words(words_list):
    vowels = "aeiouAEIOU"


def main():
  print(contains_mostly_vowel_ending_words(['file',
  'barrel', 'like', 'shoo', 'sh', 'pi']))
  print(contains_mostly_vowel_ending_words(['file',
  'barre', 'like', 'shoo', 'so', 'pi']))
  print(contains_mostly_vowel_ending_words(['do',
  'he', 'in', 'go', 'to', 'it']))
main()
```

```
False
True
True
```

Question 10

Complete the **get_unusual_average(numbers_list)** function which is passed a list of integers as a parameter. The function **returns** the average (rounded to the nearest whole number) of the numbers greater than zero in the parameter list **until the first element of the list which is greater than 100 is reached**, e.g. if the parameter list is [**4, -8, 6**, 120, -3, 20, 30] then the function returns 5, the average of 4 and 6 rounded to the nearest whole number.

Notes. 1. If there is no number in the list which is greater than 100, the function returns the average of all the positive numbers in the list.

2. If there are no positive numbers in the list, the function returns zero.

3. If there are no positive numbers in the list before an element greater than 100, the function returns zero.

```
def get_unusual_average(numbers_list):


def main():
    result = get_unusual_average([5, 10, 15, 120, 2, 88, 22])
    print(result)
    print(get_unusual_average([20, 100, 60]))
    print(get_unusual_average([-66, -100, 800, 20, 60]))
main()
```

```
10
60
0
```

Question 11

Complete the **check_password(password)** function that takes a single string parameter - a password. The function checks the password and **returns** *True* if the password is valid and *False* otherwise. For a password to be valid it needs to meet the following criteria:

- Has to be at least 8 characters in length.
- Has to have at least one alphabetical character.
- Has to have at least one numerical digit.
- Has to have at least one of the following symbols: . ; ! * ?

A few examples of the function being used are shown below. Hint: you may find the string methods **isdigit()** and **isalpha()** useful.

```
def check_password(password):
    symbol_list = [".",";","!","*","?"]

def main():
   password = "abc012"
   print("Is",password,"valid:",check_password(password))
   password = "abcd0123"
   print("Is",password,"valid:",check_password(password))
main()
```

```
Is abc012 valid: False
Is abcd0123 valid: False
```

Question 12

Complete the function **fiddle_string(text)** that takes a single string parameter `text`, and **returns** a modified version of it. How the parameter `text` is modified depends on how many characters it has.

- If the parameter `text` has an odd number of characters, then the characters before the middle character and the characters after the middle character are swapped. The middle character remains in the same place.
- If the parameter `text` has an even number of characters, then the modified string is created by swapping the first half of the string with the second half of the string.

Question 12

You can assume that the parameter `text` will have at least 1 character. A couple of examples of the function being used are shown below:

```
def fiddle_string(text):


def main():
   word = "Damir"
   print("Original word:",word,
   "\nNew word:",fiddle_string(word))
   word = "barbeque"
   print("Original word:",word,
   "\nNew word:",fiddle_string(word))
main()
```

```
Original word: Damir
New word: irmDa
Original word: barbeque
New word: equebarb
```

Question 13

Complete the **update_guess(answer, guess, character)** function that takes three string parameters. The first two parameters are **answer** and **guess**. They have the same number of characters. The third parameter character is a **string** consisting of a single alphabetical character.

The **update_guess()** function updates the **guess** string so that any instances of character in **answer** are revealed in the appropriate location in **guess**. The **update_guess()** function returns the updated **guess** string.

Question 13

A couple of examples of the **update_guess()** function being used are shown below:

```
def update_guess(answer, guess, character):


def main():
    answer = "ladder"
    guess = "******"
    guess = update_guess(answer, guess, "d")
    print(guess)
    guess = update_guess(answer, guess, "e")
    print(guess)

    answer = "explosion"
    guess = "#########"
    guess = update_guess(answer,guess,"o")
    print(guess)
    guess = update_guess(answer,guess,"x")
    print(guess)
main()
```

```
**dd**
**dde*
####o##o#
#x##o##o#
```

Question 14

Complete the **process_code(code_str)** function that takes a single string parameter **code_str**. The parameter **code_str** consists of numerical characters arranged in a specific order within which several numbers are hidden. The **process_code()** function should add these numbers and **return** their total.

The parameter **code_str** consists of several sequences of a single digit, between 1 and 5 inclusive, specifying the number of subsequent digits that make up one of the numbers we are interested in. For example if **code_str** was "**1**2**3**456" then we have the sequences **1**2 and **3**456. The numbers we are interested in are thus 2 and 456, and the **process_code()** function would return 458.

You can assume that a valid code string will always be passed as a parameter.

Question 14

A couple of examples can be seen below:

```
def process_code(code_str):



def main():
   code = "123456"
   print("Code:", code, "Result:", process_code(code))
   code = "245590000"
   print("Code:", code, "Result:", process_code(code))
main()
```

```
Code: 123456 Result: 458
Code: 245590000 Result: 90045
```