Lecture 5: Functions

Dr Simon D'Alfonso

School of Computing and Information Systems
Faculty of Engineering and Information Technology

Solutions to Lecture 4 Challenges

Write a program to:

- Devowel a given string and print the result. Devowelling is the process of removing vowels (a, e, i, o, u) from a string.
- Determine whether a given string contains a double consecutive occurrence of any characters (e.g., ‘ee’, ‘oo’, ‘11’)

Notes

An update on the current plans for the Mid-Semester Test (MST) and Assignment 1 (A1):

- A1 will be released via Grok on the Friday at the end of Week 6 (April 8) and will be due 2 weeks after that.
- The MST will be held during the lecture period of Week 7. It will be conducted online via LMS. I will put up some sample questions soon.

No need to memorize ASCII table.

Student Representatives

- Zandalee Xue <zandaleex@student.unimelb.edu.au>
- Dan Xiao <xiaodx@student.unimelb.edu.au>

Checking input with an extra conjunct

Hi Simon,

For the solution of "A and an" in Grok Worksheet 4, I was wondering why the "and" operator is needed.

```
phrase = input("Enter a phrase: ")
if phrase and phrase[0].lower() in 'aeiou':
    print("an", phrase)
else:
    print("a", phrase)
```

Logical Operators and Commutativity

- A binary operator is commutative when it gives the same results irrespective of the order of the arguments:
  - 7 + 5 = 5 + 7 (Commutative)
  - 7 – 5 ≠ 5 – 7 (Not Commutative)
- In logic, the operators *and* and *or* are commutative:
  - A and B = B and A
  - A or B = B or A

Logical Operators and Commutativity

- With the way that Python processes these logical operators, they are generally commutative, but sometimes they are not.
- Take these two examples, and swap the *and* arguments for each one:

```
my_string = "abc"
if my_string and my_string[0].islower():
    print("OK")

my_string = ""
if my_string and my_string[0].islower():
    print("OK")
```

Lecture Overview

- What are functions and why should we use them
- How to define functions, arguments and return statements
- Variables and scope

Functions

What's a function?

- takes zero or more input values, performs some task(s), and optionally returns a value
- you have already seen and used a bunch of functions by this stage, such as ord(), int(), str(), len(), round() ...
- Wouldn’t it be nice to be able to recycle chunks of our own code?

User-defined Functions

In order to define a function, we need:

- A function name (following same conventions as variable names)
- (optionally) a list of input variables (arguments)
- (optionally) an output object (via return)

```
def <function_name> (<argument(s)>):
    statement block
```

Why define functions?

Defining our own functions means:

- We cut down on repeated code
- Nice function names makes our code clear and easy to read
- If it's a sub-task, create a function
- We can move bulky code out of the way
- Promotes code that is easier to understand as details are hidden inside functions
- Modularity

Simple Example

```
def my_function():
    print('This is a function')

my_function()
```

Examples with 1 argument

```
#Print the number of digits in a number
def print_digits(n):
    s = str(abs(n))
    print(len(s) - ('.' in s))

>>> print_digits(343.12)
5

#Convert from Celsius to Fahrenheit and print the
result:
def C2F(n):
    print(9 * (n / 5) + 32)

>>> C2F(24)
75.2
```

Example with 2 arguments

```
def area_triangle(base, height)
    return 0.5 * base * height
```

Functions: More details

It is possible to define functions which take a variable number of arguments. This is done by specifying default values for arguments

```
def seconds_in_year(days = 365):
    print(days*24*60*60)

>>> seconds_in_year()
31536000
>>> seconds_in_year(366) #leap year
31622400
```

Functions: the power of returning

In order to use the output of a function from where it was called, we need to *return* a value:

```
#Convert from Celsius to Fahrenheit:
def C2F(n):
    return(9 * (n / 5) + 32)

cel = 21
fah = C2F(cel)

#Print the number of digits in a number
def count_digits(n):
    s = str(abs(n))
    return(len(s) - ('.' in s))

print(f"No. digits is: {count_digits(-123.123)}")
```

Returning Boolean

```
def is_even(num):
    if num % 2 == 0:
        return True
    else:
        return False
```

Returning None

```
def happy_birthday(name):
    print("Happy birthday,", name)

x = happy_birthday("Harry")
print(x)
```

The output of the program above is:

Happy birthday, Harry
None

First, the print statement within the function is executed. Second, if a function does not contain a return statement, it returns None by default, thus None is assigned to x and printed. The following explicit inclusion of return None achieves the same function:

```
def happy_birthday(name):
    print("Happy birthday,", name)
    return None
```

Returning Multiple Values

Multiple values can be returned by separating the outputs with a comma:

```
def multiple_return_values():
    return 'a', 1

response = multiple_return_values()
print(response) #displays ('a', 1), which is a tuple
```

Tuples are another type of sequence that we will cover later in the semester. Like strings and lists, they can be accessed with the subscript operator.

```
print(response[0]) #outputs a
print(response[1]) #outputs 1
```

Exercise 1

Write a function that takes in two strings as input, and returns a count of the number of characters they have in common.

Exercise 1 Solution

```
def common_characters_count(string1, string2):

    common_characters_count = 0;

    string1_unique_chars = ""
    for char1 in string1:
        if char1 not in string1_unique_chars:
            string1_unique_chars = string1_unique_chars + char1

    for char1 in string1_unique_chars:
        if char1 in string2:
            common_characters_count = common_characters_count + 1

    print(string1 + " and " + string2 + " have " +
str(common_characters_count) + " characters in common")



>>> #testing the function
>>> common_characters_count("hello", "goodbye")
hello and goodbye have 2 characters in common
```

Recap on function arguments

- Functions can have 0 or more input arguments.
- Arguments can be required or made optional, by setting a default value.

```
def greet(name, message='Hi'):
    print(message, name)

>>> greet("Sally")
Hi Sally
>>> greet("Sally", "Welcome")
Welcome Sally
>>> greet()
?
```

Functions: Positional Arguments

When we call a function that requires multiple arguments order matters. The most common way to send arguments to a function is by their position or order.

```
def print_person_name(title, first_name, last_name):
    print(title, first_name, last_name)

>>> print_person_name("Mr", "Guido", "van Rossum")
Mr Guido van Rossum
```

Functions: Keyword arguments

Though positional arguments are the most common way to send arguments to a function they are not the only way. You can use the name of the argument and explicitly assign each argument a value in any order.

```
def print_person_name(title, first_name, last_name):
    print(title, first_name, last_name)

>>> print_person_name(last_name="van Rossum",
title="Mr", first_name="Guido")
Mr Guido van Rossum
```

Functions - Returning early

If your function has the answer it needs, you can return straight away.

```
"""
Returns True if string contains
the character 'x', False
otherwise.
(Inefficient way)
"""
def has_x(string):
    has_x = False
    for char in string:
        if char == 'x':
            has_x = True
    return(has_x)
```

```
"""
Returns True if string contains
the character 'x', False
otherwise.
(Smart way)
"""
def has_x(string):
    for char in string:
        if char == 'x':
            #why wait?
            return(True)
    return(False)
```

Exercise 2

Write a function that takes two arguments: a block of text and a word. The function should return True if the word is found in the text, otherwise False.

Note: only exact matches should return True. For example, "sub" is not found exactly in "There are substrings".

Exercise 2 Solution

```
def is_word_in_text(word, text):
    if ' ' + word + ' ' in text:
        return True

    if word + ' ' in text and text.startswith(word):
        return True

    if ' ' + word in text and text.endswith(word):
        return True

    return False
```

Functions and Methods

Some clarification on terminology.

- Functions and methods provide pre-defined functionality over a pre-defined set of arguments (generally, of fixed type), in the form of a predefined set of outputs.
- Functions share the same namespace as variables and are called as “standalones”.

```
>>> type(len)
<class 'builtin_function_or_method'>
```

Functions and Methods

Unlike standalone functions, methods are defined for/called from objects of a given type and are called as object.METHOD() from objects of that type.

```
>>> type(upper)
NameError: name 'upper' is not defined

>>> "a piece of string".upper()
'A PIECE OF STRING'
```

Variables and Scope

Each function (call) defines its own local variable "scope". Its variables are not accessible from outside the function (call).

```
def subtract_one(k):
    k = k - 1
    return k

i = 0
n = subtract_one(i)
print(i)
print(n)
print(k) #k has not been defined at this level
```

Everything modular

You might sometimes see the main level of code placed within a main() function.

```
def subtract_one(k):
    k = k - 1
    return k

def main():
    i = 0
    n = subtract_one(i)
    print(i)
    print(n)

main()
```

Variables and Scope

```
def subtract_one(i):
    i = i - 1
    return i

i = 0
n = subtract_one(i)
print(i)
print(n)
```

What is the output?

Variables and Scope

Functions can access variables defined outside functions ("global" variables), although they should be used with extreme caution.

```
def fun1(j):
    fun2(j)
    return 1

def fun2(k):
    global i,j # global variables
    print(i,j,k)
    return 2

i,j,k = 1,2,3
fun1(i) #what does this output?
```

Summary

Today we covered:

- The structure for creating functions
- Function arguments and return values
- Using arguments: optional arguments, positional and keyword
- Returning values from functions including the efficiency of returning early
- Variables and scope

Lecture 5 Challenges

1. Write a function ascii_match(), which accepts two required input arguments, an integer and a string. If the ASCII numbers of all the characters in the string add up to the value of the input integer, return True. Otherwise, return False.
2. Write a function avg_three() that returns the average (mean) of three integers between 1 and 9. The function should allow for three optional input arguments (num1, num2, num3), but if any of these arguments is not provided when the function is called, then a random number is generated for that value. See https://www.w3schools.com/python/ref_random_randint.asp for how to generate random integers.

Lecture Identification and Acknowledgement

Coordinator / Lecturer: Simon D'Alfonso

Semester: Semester 1, 2022

These slides include materials from 2020 - 2021 instances of COMP90059 run by Kylie McColl or Wally Smith