CS5710 :- Lab 2

Stephen Piddock

January 24, 2024

1 Learning outcomes

This lab session will introduce you to

- working with Python functions and string manipulation.
- Writing Python programs of varying difficulty.

2 Notes

- Complete the various sections described in this document. and have a look at the extensions.
- **Remember** the TAs and lecturers are here to help - if you are stuck then don't hesitate to ask questions.
- Read carefully the instructions, and ask for help if you feel lost.
- It is recommended that you store your programs in a folder hierarchy comprising of a single high-level folder, e.g., `CS5710Labs`, and one sub-folder for each lab session, e.g., `lab2`, `lab3`, etc.
- Unless stated otherwise, store your programs in the files called `ex<exercise_number>.py`. For example, the program for Exercise 1 should be stored in file `ex1.py`.
- If you work on a lab workstation, use the `Y:` folder to store your work as everything stored in the local folders is being erased on a daily basis.
- If your program gets stuck in an infinite loop, you can interrupt it with `CTRL-C` or click on the red box above the console window in Spyder.

3 Strings and loops over strings

Note - don't delete the code you write in this section as you will reuse it later!

3.1 Write a program that asks the user for a string and prints its first and last character. If the string has length one, it should just print one character. If the string is empty, is should just print a new line.
Hints:

- Use the built-in function `len` to determine the length of a string. For example, `len('abc')` evaluates to 3.
- To determine whether a string is empty, you can either compare its length with 0, or compare it directly against `''`, which represents the empty string.

3.2 Write a program that asks the user to enter a string. If the string contains at least one of every vowel (`a`, `e`, `i`, `o`, `u`), print `'All vowels are there!'`. Additionally, if the string starts with the letter `a` and ends with the letter `z`, print `'And it could be alphabetical!'`.
Hint: use `in` to determine if a given string is contained in another string. For example, `'a' in 'xyab'` evaluates to `True`.

3.3 Write a program that asks the user to enter a string, and returns the number of vowels (`a`, `e`, `i`, `o`, `u`) appearing at all odd indices. For example, if the user enters `'eggs over easy'`, then your program must output 3.

3.4 Write a program that asks the user to enter a string, and using string slicing determines whether the string is a concatenation of three identical strings. If so, print `'string is a triple concat of '` followed by the relevant substring; otherwise, print `'string is not a triple concat'`. For example, if the user enters `'hello,hello,hello,'`, the output must be `'string is a triple concat of hello,'`.

3.5 Write a program that asks the user to enter a string `s`, and outputs the number of times the string `'tart'` appears in `s`. For example, if the input is `iliketartartandmoretart`, the output must be 3. Note that simply calling `s.count('tart')` may not produce a correct answer as it only counts the number of *non-overlapping* substrings. That is, in the above example, `'iliketartartandmoretart'.count('tart')` evaluates to 2 since the second occurrence of `tart` overlaps the first one.

4 Simple functions

Note - make sure you write a docstring for each of these functions.

4.1 Write a function `max3()` that takes three integers as input arguments and returns the largest of them. Include a docstring describing the inputs, outputs, and what is being computed.

4.2 Write a function `sort3()` that takes three integers as input arguments and returns a tuple consisting of the input numbers sorted in the increasing order of their values. For example, `sort3(120, 3, 23)` should return `(3, 23, 120)`.

4.3 Write a function `min_max()` that takes a string of digits as input and returns a string consisting of the smallest and the largest integers in the input string. Do this by iterating over the elements of the string using the `for`...`in` loop. Do not use the built-in `min()` and `max()` functions. For example, `min_max("31805")` should return `"08"`.

4.4 Write a function `reverse()` that takes one argument that can be either a string as input, and returns its reversal. For example, `reverse("left to right")` should return the string `"thgir ot tfel"`.

4.5 A *palindrome* is a string that is the same as its reversal. Using `reverse()` write the function `partialPalindrome` that takes a string `s` and an integer `n` as input. `partialPalindrome` returns a boolean that is `True` is there is a sub-string in `s` that is a palindrome of length `n` and `False` otherwise.

For example, `partialPalindrome("AGTTGCC",4)` will return `True` and `partialPalindrome("AGTTGCC",3)`, will return `False`.

4.6 Write a function `extract_strings` that takes three arguments with arbitrary types, and returns a string with all the arguments that are strings concatenated together. For example, `extract_strings(('foo', 5, 'aaa'))` should return `('fooaaa')`.
Hints:

- Use `type(x)` to determine the type of variable `x`.
- The keyword to denote the string type in Python is `str`.

4.7 Rewrite the code you wrote in the first section on strings so that they are functions. Each function should have a single argument which is a string. The function should not read in a string.

Write a new function `runThis` which accepts three arguments, a string `s` and functions `fa` and `fb`. `runThis` should check if the type `fa` and `fb` returns is the same and returns an integer or string. If that is true then it should return `fa(s) + fb(s)`. If it is false then it should return `None`. Hints:

- The keyword to denote the integer type in Python is `int`.

5 Advanced functions

In this exercise, you will practice implementing generic functions by means of passing functions as arguments to other functions. Copy the following code into Spyder, and fill the missing parts as per the text in the comments:

```
def area(shape, n):
#    replace pass with a line to return the area
#    of a generic shape with a parameter of n
    pass

def circle(radius):
    """
    Input: radius, a positive float
    Returns the area of a circle with the given radius
    """
    return 3.14*radius**2

def square(length):
    """
    Input: length, a positive float
    Returns the area of a square with sides of the given length
    """
    return length*length

print(area(circle,5))  # example function call
```

5.1 Write expressions that use `area()` to calculate the following quantities:

(a) the area of a circle with a radius of 10;

(b) the area of a square with sides of length 5;

(c) the area of a circle with *diameter* of length 4.

5.2 In this exercise, you will practice function nesting as well as returning a function from another function. Note how nesting functions inside other functions allows you to compartmentalize functionalities that are relevant only in the context of specific tasks, as opposed to the program as a whole. Also, note how the variable visibility is affected by the nesting.

Copy the following code into Spyder, and fill the missing parts as per the text in the comments:

```
def person(age):
    print("I am a person")
    def student(degree):
        print("I like learning")
        def holiday(place):
            print("But I need to take holidays")
            print(age,"|",degree,"|",place)
        # write a line to return the appropriate function
    # write a line to return the appropriate function
```

For example, the function call

```
person(12)("Math")("beach")  # example function call
```

should print this:

```
I am a person
I like learning
But I need to take holidays
12 | Math | beach
```

(a) Write a function call with age of 29, degree of "Data Science", and vacation place of "Japan".

(b) Write a function call so that the last line of its printout is as follows:

```
23 | "Law" | "Barbados"
```