

ITI 1120 - Lab # 4

Fall 2022

**order of execution, more strings, for
loops, range function**

What is in this lab?

1. This lab has 6 Tasks . You should try to do these parts at home.
2. 5 programming exercises (0 to 4, for your Lab participation)

The slides that are included here but do not explain either Tasks or Programming exercises are there as reminder of some relevant material that is needed for this lab.

Lab 4 overview

- Open a browser and log into Brightspace
- On the left hand side under Labs tab, find lab4 material contained in [lab4-students.zip](#)
- Download that file to the Desktop and open it.
- Read slides one by one and follow the instructions explaining what to do. When stuck ask for help from the TA. Ideally you should try to complete some, or all of this, at home and then use the lab as a place where you can get help with things you may have had difficulties with.

Task 0

In a code cell, assign string `'good'` to variable `s1`, `'bad'` to variable `s2` and `'silly'` to variable `s3`. Write Python expressions involving strings `s1`, `s2`, and `s3` that correspond to (translate the following human language to Python expression):

- a) `'ll'` appears in `s3`
- b) the blank space does not appear in `s1`
- c) the concatenation of `s1`, `s2`, and `s3`
- d) the blank space appears in the concatenation of `s1`, `s2`, and `s3`
- e) the concatenation of 10 copies of `s3`
- f) the total number of characters in the concatenation of `s1`, `s2`, and `s3`

Indexing operator, revisited

The indexing operator returns the character at index *i* (as a single character string).

S	=	'	A	p	p	l	e	'
			0	1	2	3	4	
s[0:2]	=	'	A	p	'			
s[1:4]	=		'	p	p	l	'	
s[2:5]	=			'	p	l	e	'
S[2:]	=			'	p	l	e	'
s[:2]	=	'	A	p	'			
s[3:1]	=			'	p	l	'	

s[i:j] : the slice of **s** starting at index **i** and ending **before** index **j**

s[i:] : the slice of **s** starting at index **i**

s[:j] : the slice of **s** ending before index **j**

```
>>> s= 'Apple'
>>> s[0:2]
'Ap'
>>> s[1:4]
'ppl'
>>> s[2:5]
'ple'
>>> s[2:]
'ple'
>>> s[:2]
'Ap'
>>> s[-3:-1]
'pl'
```

Task 1

In the notebook provided

1. In code cell, create `aha` variable and assign `'abcdefgh'` to it.

2. Then write Python expressions (in the code cell) using string `aha` and the indexing/slicing operator that evaluate to:

- a) `'abcd'`
- b) `'def'`
- c) `'h'`
- d) `'fg'`
- e) `'defgh'`
- f) `'fgh'`
- g) `'adg'`
- h) `'be'`

String methods

Strings are
immutable;
none of the
string methods
modify string **s**

Usage	Explanation
<code>s.capitalize()</code>	returns a copy of s with first character capitalized
<code>s.count(target)</code>	returns the number of occurrences of target in s
<code>s.find(target)</code>	returns the index of the first occurrence of target in s
<code>s.lower()</code>	returns lowercase copy of s
<code>s.replace(old, new)</code>	returns copy of s with every occurrence of old replaced with new
<code>s.split(sep)</code>	returns list of substrings of s , delimited by sep
<code>s.strip()</code>	returns copy of s without leading and trailing whitespace
<code>s.upper()</code>	returns uppercase copy of s

Task 2

Run the first cell of Task 2 in colab notebook, it defines the following string:

```
s = '''It was the best of times, it was the worst of times;  
it was the age of wisdom, it was the age of foolishness;  
it was the epoch of belief, it was the epoch of incredulity;  
it was ...'''
```

(The beginning of A Tale of Two Cities by Charles Dickens.)

Then do the following, in order:

- (a) Write a sequence of statements that produce a copy of `s`, named `newS`, in which characters `.`, `,`, `;`, and `\n` have been replaced by blank spaces.
- (b) Remove leading and trailing blank spaces in `newS` (and name the new string `newS`).
- (c) Make all the characters in `newS` lowercase (and name the new string `newS`).
- (d) Compute the number of occurrences in `newS` of string `'it was'`.
- (e) Change every occurrence of `was` to `is` (and name the new string `newS`).

Task 3

1. Use Python Visualizer (the following url) to run the following code snippets (found in the notebook). Determine when to use *print* and when to use *return*

<http://www.pythontutor.com/visualize.html#mode=edit>

Task 4

Run each cell in within Task4's section.

Compare the result and think about why it is the case.

Examples of for loops with range ()

```
>>> for i in range(4):  
    print(i)
```

```
0  
1  
2  
3  
>>>
```

```
>>> for i in range(1):  
    print(i)
```

```
0  
>>>
```

```
>>> for i in range(0):  
    print(i)
```

```
>>>
```

```
>>> for i in range(2, 3):  
    print(i)
```

```
2  
>>>
```

```
>>> for i in range(2, 6):  
    print(i)
```

```
2  
3  
4  
5  
>>>
```

```
>>> for i in range(2, 2):  
    print(i)
```

```
>>>
```

```
>>> for i in range(0, 16, 4):  
    print(i)
```

```
0  
4  
8  
12  
>>>
```

```
>>> for i in range(2, 16, 10):  
    print(i)
```

```
2  
12  
>>>
```

Task 5

Before attempting the following exercise, study the examples from the previous slide.

Then add new code cells in the ipython notebook each cell produce one of the following outputs by a loop.

Note that if you put `,end=" "` at the end of a `print` function call (as an argument), then the print function will print the blank space when it finishes rather than go to the new line.

Eg: this prints numbers 0 to 9 in one line :

```
>>> for i in range(10):  
print(i,end=" ")  
0 1 2 3 4 5 6 7 8 9 >>>
```

a) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

b) 1, 2, 3, 4, 5, 6, 7, 8, 9

c) 0, 2, 4, 6, 8

d) 1, 3, 5, 7, 9

e) 20, 30, 40, 50, 60

f) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

Programming Exercise 0

Repeat the exercise in **the task 3 (version 2)**, where in addition you need to ask the user for their citizenship and if they are currently in prison convicted for a criminal offence. Your program should print a nice message telling the user if they are eligible to vote (i.e. if they are 18+, Canadian and do not live in prison convicted for a criminal offence, then they can vote. Otherwise not). You should modify function **is_eligible** so it takes to additional parameters as input. In particular, the definition of the function should be:

is_eligible(age, citizenship, prison)

You should define your function in a cell and your I/O(input and output, user interactions) in another cell

Your program should work if the user enters any of the following versions of answers for the two new questions:

```
Canadian  
Canada  
canada  
canadian  
Yes  
YES  
No  
no
```

Note that in Canada, one can vote even if in prison convicted for a criminal offence. This example is fictional.

Programming Exercise 1

Write a function called `mess` that takes a phrase (i.e., a string) as input and then returns the copy of that phrase where each character that is one of the last 8 consonants of English alphabet is capitalized (so, r, s, t, v, w, x, y, z) and where each blank space is replaced by dash.

For this question, use a `for` loop over characters of a string, and “accumulator”. (We have seen that in Lecture 6). When called from the python shell, your function should behave as follows:

```
>>> mess('Random access memory ')
'Random-acceSS-memoRY--'
>>> mess('central processing unit.')
'ceNTRal-pRoceSSing---uniT.'
```

Built-in function `range()`

Function `range()` is used to iterate over a sequence of numbers in a specified range

- This iterates over the `n` numbers `0, 1, 2, ..., n-1`

```
for i in range(n):
```

- This iterates over the `n` numbers `k, k+1, k+2, ..., n-1`

```
for i in range(k, n):
```

- This iterates over the `n` numbers `k, k+c, k+2c, k+3c, ..., n-1`

```
for i in range(k, n, c):
```

In particular, the first time a program encounters a for-loop it **creates the variable** whose **name** follows the keyword **for**. In the above examples, the variable name is **i**. Then that variable, **i** in this case, takes values in the given range one by one. Each time it takes the next value it enters the for-loop and executes its body. The for-loop terminates after **i** has taken on all the values in the range, as shown above)

Python Visualizer

Go to Python Visualizer here (make sure you choose Python 3)

<http://www.pythontutor.com/visualize.html#mode=edit>

and copy/paste, only by one, the following loops to it and click Forward to visualize the execution. Pay attention what is assigned to variable **i** and when does loop terminate.

```
for i in range(3):  
    print(i)
```

```
for i in range(2,4):  
    print(i)
```

```
for i in range(2,2):  
    print(i)
```

```
for i in range(1,10, 3):  
    print(i)
```


Programming Exercise 2

Use the function `is_divisible23n8()` which is already given,

1. write a function called `print_all_23n8(num)`, that takes as input a non-negative integer `num` and prints all the non-negative numbers smaller than `num` that are divisible by 2 or 3 but not 8. You should use the function that is already present in the file (and that you developed for the last lab).
2. Outside of that function ask the user for a non-negative integer. Your program should then print all non-negative numbers that are divisible by 2 or 3 but not 8, by making a call to function `print_all_23n8(num)`
3. Run your program and test it by entering, for example, 1000 when prompted for a number

Programming Exercise 3

This program:

```
for i in range(4):  
    print("*****")
```

Prints :

* * * * *
 * * * * *
 * * * * *
 * * * * *

Write a program that asks a user for a positive integer and a character. And then draws half pyramid with the given number of rows using that character. For example, if the user enters 3 and \$, your program should draw (that is, print):

\$
\$ \$
\$ \$ \$

Bonus exercise:

For a challenge, draw a real pyramid
like the one to the right
that should be displayed if the user entered 10 and #

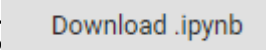
```
#  
##  
###  
####  
#####  
#####  
#####  
#####  
#####  
#####  
#####
```

Programming Exercise 4

1. Write a program that asks a user for a **positive integer** and then prints all the **divisors** of the given integer. For example, if the user entered 6, the program should print: 1, 2, 3, 6
2. Add a function, called **prime**, to this program. Function **prime** takes a **positive integer** as input parameter and tests if it is a **prime** (that is, it returns true if the given number is a prime and false otherwise). Recall that a number is prime if it at least 2 and if it is only divisible by 1 and itself. Then **make a call to this function** and print the message stating if the number the user entered in 1 is a prime.
3. Copy/paste your whole solution into Python Visualizer, click through with Forward to understand see how it runs.
4. **Bonus exercise:** Write a program that asks a user for a positive integer, n , and prints all the primes smaller than n .

How to submit lab4:

You need to submit your Lab4_Exercises **notebook** (ipynb file) to Brightspace (Under the Assignments Tab) after you finish the Lab and get graded from the TA .

You can download your ipynb file using  in colab from the file menu.

Also, please **do not compress** your files (no zip or rar). Submit both **individually**.