ITI 1120

Lab # 3

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

Starting Lab 3

- Open a browser and log into Brightspace
- On the left hand side under Labs tab, find lab4 material contained in lab3-students.zip file
- Download that file to the Desktop and unzip it.

Before starting, always make sure you are running Python 3

This slide is applicable to all labs, exercises, assignments … etc

ALWAYS MAKE SURE FIRST that you are running Python 3

That is, when you click on IDLE (or start python any other way) look at the first line that the Python shell displays. It should say Python 3 (and then some extra digits)

If you do not know how to do this, read the material provided with Lab 1. It explains it step by step

Task 1

In Python interpreter 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:

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

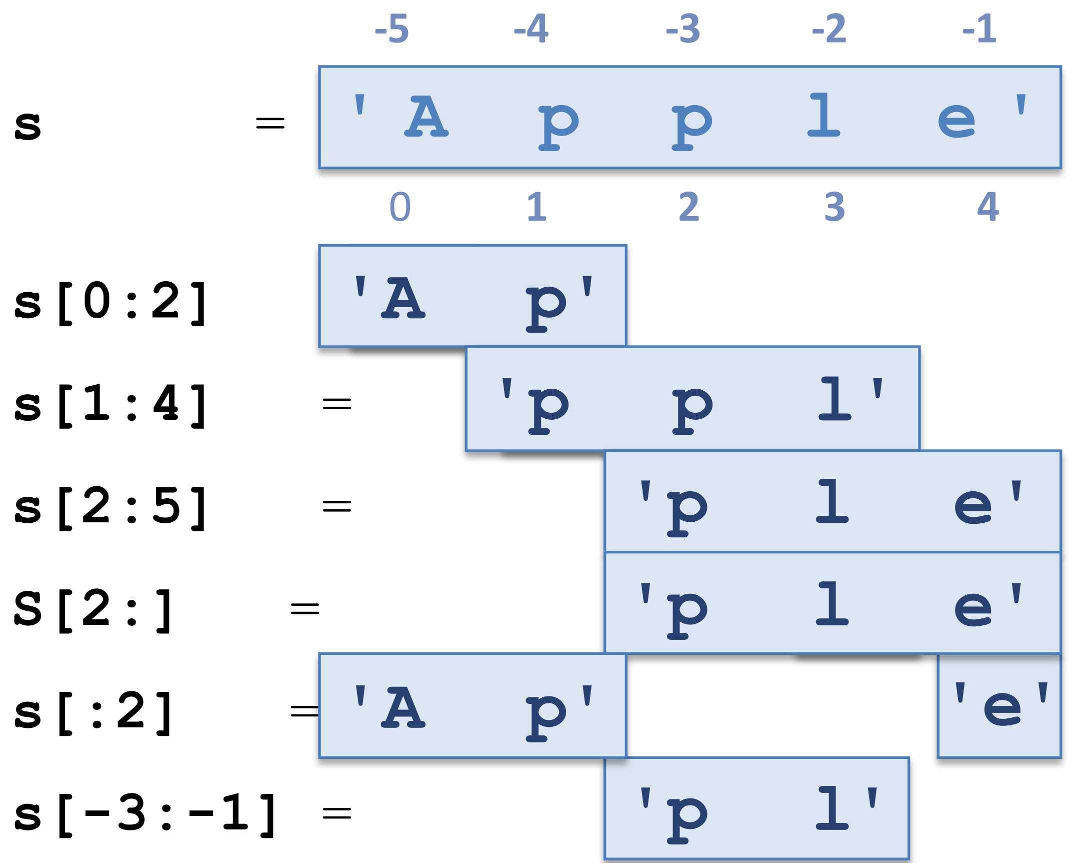
Indexing operator, revisited

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

`s[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`



Note: There are no blank spaces here, just appears so for visibility purposes

```
>>> 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 2

Start python interpreter

1. In python interpreter, create **aha** variable and assign **‘abcdefgh’** to it.

2. Then write Python expressions (in the interpreter) using string **aha** and the indexing 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
`s.capitalize()`	returns a copy of `s` with first character capitalized
`s.count(target)`	returns the number of occurences of `target` in `s`
`s.find(target)`	returns the index of the first occurrence of `target` in `s`
`s.lower()`	returns lowercase copy of `s`
`s.replace(old, new)`	returns copy of `s` with every occurrence of `old` replaced with `new`
`s.split(sep)`	returns list of substrings of `s`, delimited by `sep`
`s.strip()`	returns copy of `s` without leading and trailing whitespace
`s.upper()`	returns uppercase copy of `s`

Task 3

Copy/paste the following expression (in black) to Python Interpreter:

```
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 4

a) Follow the link in (b) and trace through the two Python Vizualizer examples that have "Code Lence 1" and "Code Lence 2" in the caption. You do that by clicking Forward until the program ends, like we did it class (It does not matter here that it says Python 2.7. For these two programs Python 2 and 3 behave the same).

b) Then Answer the two multiple choice questions at the end

https://runestone.academy/runestone/static/thinkcspy/Functions/Functionsthatreturnvalues.html

c) Then follow this link and do the two multiple choice exercises:

https://runestone.academy/runestone/static/thinkcspy/Functions/FlowofExecutionSummary.html

Task 5: More tracing and print vs return

Open the program called print-vs-return-and-function-calls.py, provided in this lab.

The lines the start with # are commented out. Thus Python will ignore them. In other words, if you press Run Module the lines that start with # will not be executed. Uncomment the lines **as instructed** in the file. Each time you do, save and press "Run Module". But before you press "Run Module", write down what you think the program will print. Then press "Run Module" and compare.

Note: once you uncomment a line as instructed, leave it as is (do not put back the comments, but rather continue with the next set of lines you are instructed to uncomment).

Programming exercise: Solved Problem

Suppose that you are given the following two problems to solve.

Read the two problems. Think about how you would solve them, and then open and study the two provided solutions in prog_solved_v1.py and prog_solved_v2.py. Run both.

Version 1: Write a program that asks a user for her name and age and prints a nice message stating if the user is eligible to vote.

Version 2: Write a program that asks a user for name and age and prints a nice message stating if the user is eligible to vote. As a part of your solution the program should have a function called is_eligible that given the age as input parameter returns true or false depending on weather the age is less than 18 or not.

Programming exercise 1

Repeat the exercise in the previous question (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 paramters as input. In particular the head of the function should be: is_eligible(age, citizenship, prison)

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
```

and so on

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

Programming exercise 2

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 will see, or have seen, that in Lecture 8 on Monday). 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)

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
>>>
```

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)
```

Task 6

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

Then open a new file in IDLE and write a program with 6 separate for loops that will print the 6 sequences listed below in parts a) to f). (you do not have to print commas, but you can if you know how to).

Note that if you put ,end=" ″ at the end of a print function call, 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 3:

Open the file ex23n8.py. Inside of that file:

1. write a function called print_all_23n8(num), that takes as input a non-negative integer num and prints all the 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

3. Run your program and test it by entering, for example, 1000 when prompted for a number

Programming exercise 4:

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 piramid with the given number of raws using that character. For example if the user enter 3 and $, your program should draw (that is, print):

```
$
$$
$$$
```

Bonus excercise:

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

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

Programming exercise 5:

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 inputted 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.

Making a table

With these tools, you now can make a program that prints nice tables. See here:

https://runestone.academy/runestone/static/thinkcspy/MoreAboutIteration/SimpleTables.html