

Name: Carlos Abreu
Email: 2023693@student.uwtsd.ac.uk
Class: Software Development
Course: Cloud Computing
Prof: Renuga Jayakumaar

Part A

A1 .

In computer programming there are two fundamental aspects: data and instructions. To work with data it's necessary to understand variables and data types, and to work with instructions, it's necessary to understand control structures and statements.

According to mathematicians and computer scientists, any computer program can be written using basic **control structures** combined in any way to solve a specified problem.

There are three basic types of control structures in programming: sequence, selection and iteration.

A sequence of statements follows from top to bottom flow in executing a program, such that the statements are executed line by line in the order in which they appear.

The selection structure is used to test a condition. Condition refers to any expression or value that returns a Boolean value, meaning true or false. Which means the program chooses between two or more alternative paths, and is executed depending whether or not the result of the condition is true or false.

The main basic types of selection are "if", "if/else" and "switch" statements. Usually, the "if" statement is the most common and it evaluates or tests the conditions, along with a statement or statements to be executed if the condition is true, and optionally, other statements to be executed if the condition is determined to be false. Either "if" or "if/else" statements can be nested. Switch statements is used when there are multiple cases to choose from, this construct are common in many others computer language, but it uses in Python is available only recently with Python 3.10 version with keyword "match".

The iteration or repetition structure also known as loop structures, repeatedly executes a series of statements as long as the condition is true. "While," "do/while" and "for" loop are the three main types of iteration statements. Repetition "For" will repeat one or two program statements set by a counter. The "do/while" statement is unknown in Python, but common in others computer language. A loop can either be controlled by event or counter. An event-controlled loop executes a sequence of statements till an event occurs and a counter-controlled loop executes the statements a predetermined number of times. [1]

Python "if/elif" statements

if expression:

statements

elif expression:

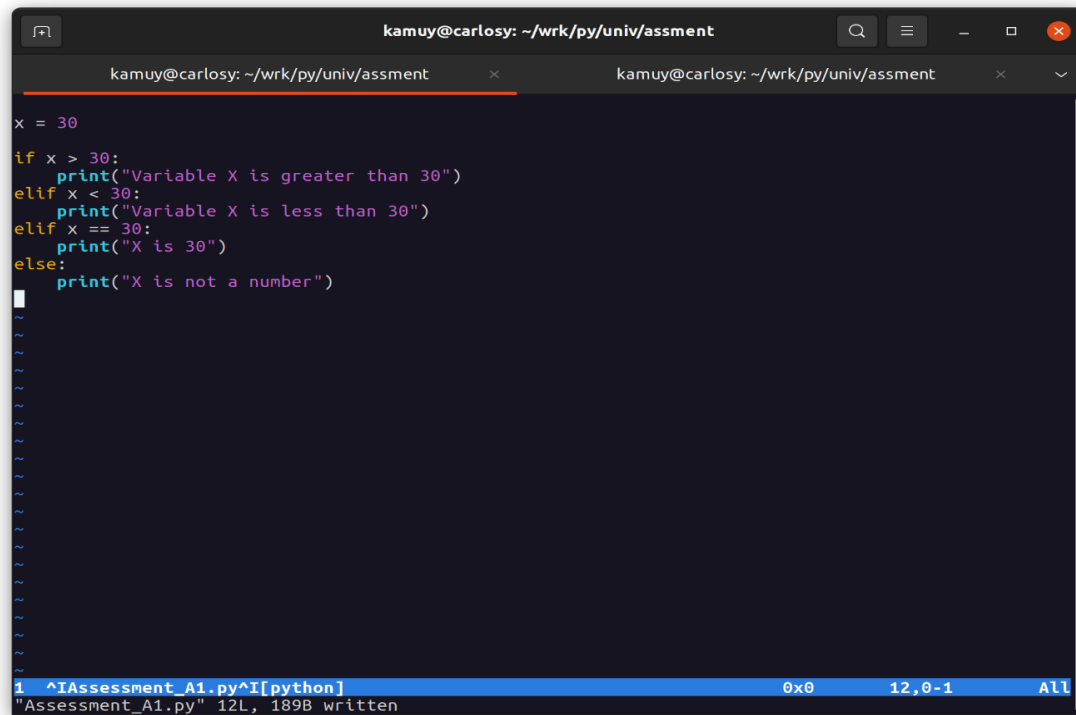
statements

else:

statements

The elif is short for else if.

Example

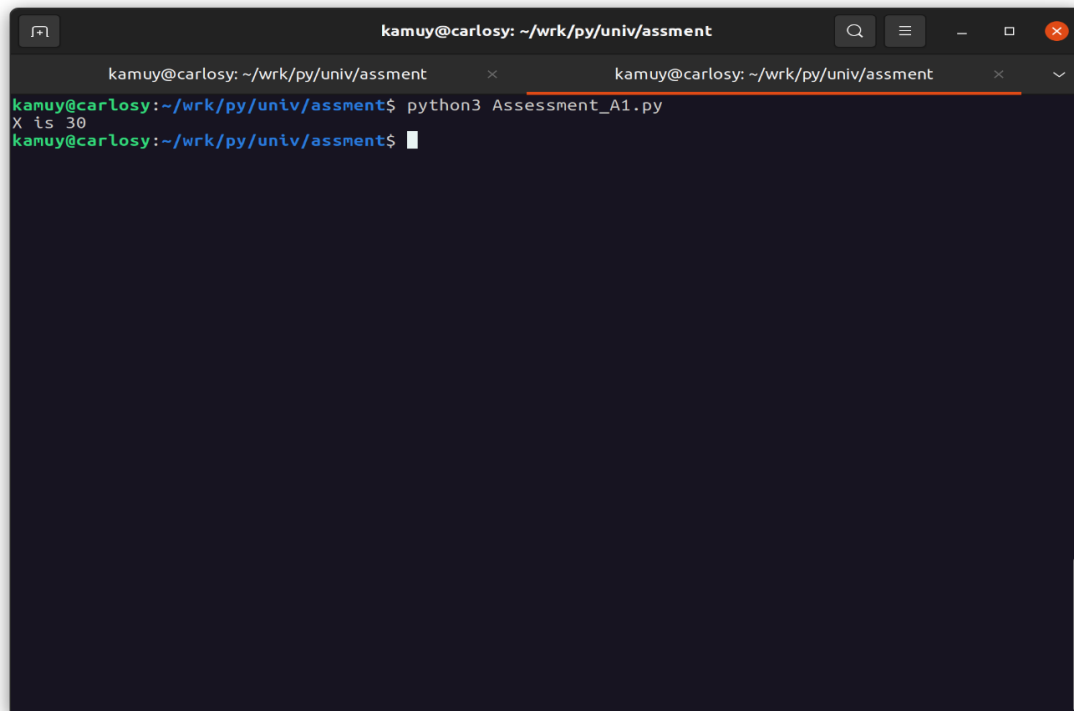


The screenshot shows a code editor window with the title "kamuy@carlosy: ~/wrk/py/univ/assment". The editor contains a Python script with the following code:

```
x = 30
if x > 30:
    print("Variable X is greater than 30")
elif x < 30:
    print("Variable X is less than 30")
elif x == 30:
    print("X is 30")
else:
    print("X is not a number")
```

The script is saved as "Assessment_A1.py" and is 12 lines long, 1898 bytes in size. The status bar at the bottom indicates the file is named "Assessment_A1.py" and is 12 lines long, 1898 bytes in size.

Output

A screenshot of a terminal window with a dark background. The window title is 'kamuy@carlosy: ~/wrk/py/univ/assment'. The terminal shows a prompt 'kamuy@carlosy:~/wrk/py/univ/assment\$' followed by the command 'python3 Assessment_A1.py'. The output of the script is 'X is 30'. The prompt is now 'kamuy@carlosy:~/wrk/py/univ/assment\$' with a cursor. There are two tabs open in the terminal window, both with the same title.

```
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A1.py
X is 30
kamuy@carlosy:~/wrk/py/univ/assment$
```

In the above scree-shoot Python evaluates each condition one by one and if a true condition is found the statement block under that condition will be executed. If no true condition is found the statement block under else will be executed. [2]

A2 .

A loop is a fundamental programming structure that is often used in writing computer programs. It is a sequence of instructions that is repeated until a certain condition is reached. There are two types of loops in Python, for and while with its variations using else clause. For loop has two sections: a header specifying the iterating conditions, and a body which is executed once per iteration. The header often declares an explicit loop counter or loop variable, which allows the body to know which iteration is being executed.

Every loop has 3 parts: initialization, condition and update (increment or decrement).

While loops repeats as long as a certain boolean condition is met.

It's structure is:

initialization

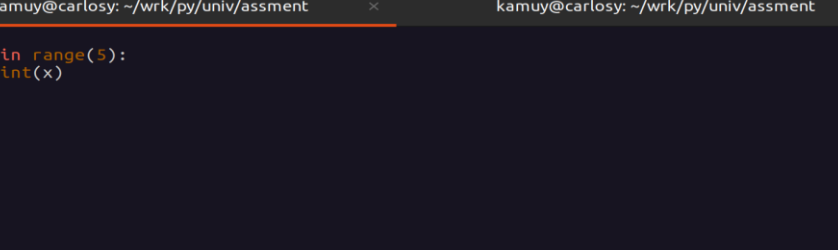
while (**condition**):

 //Code block to execute something

 // **update**

Python range() function supports up to three parameters: range(start,stop,step), and at least one parameters (stop) must be included: **[3]**

- It is important to note that all parameters must be integers and can be numbers positive or negative .



The screenshot shows a PyCharm IDE window with the title "kamuy@carlosy: ~/wrk/py/univ/assment". The editor displays a Python script:

```
1  
2 for x in range(5):  
3     print(x)
```

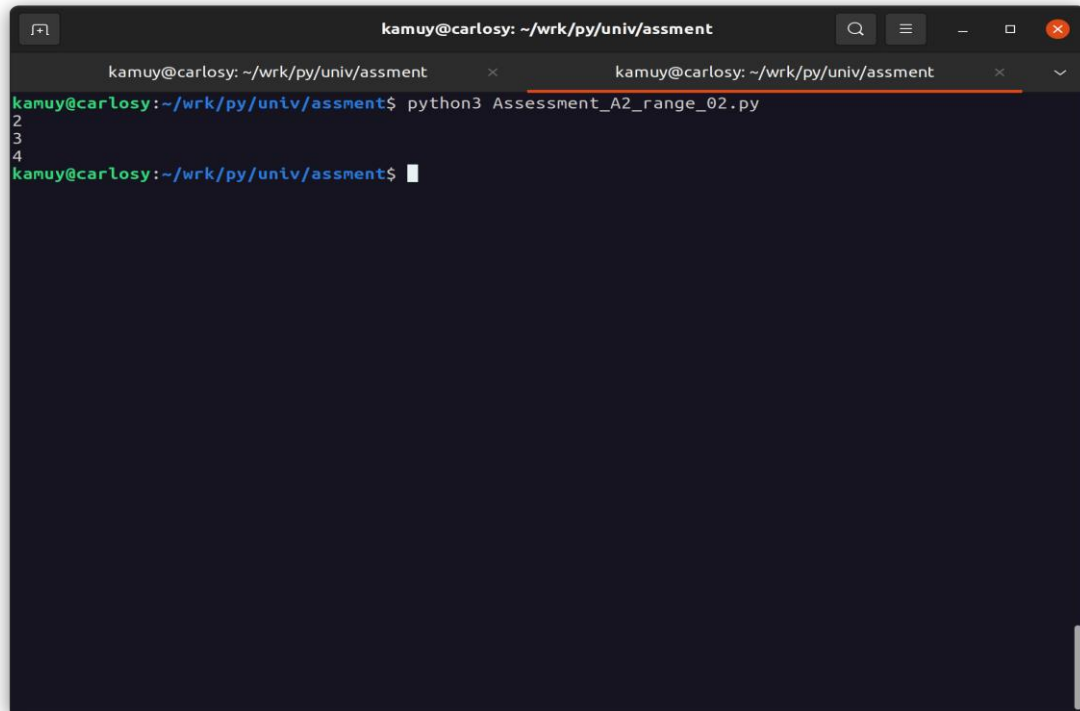
The output console at the bottom shows the execution results:

```
1 ^IAssessment_A2_range_01.py^I[python]  
"Assessment_A2_range_01.py" [New] 3L, 33B written
```

The status bar at the bottom indicates the current line is 1, column 11, and the scope is "All".

Output

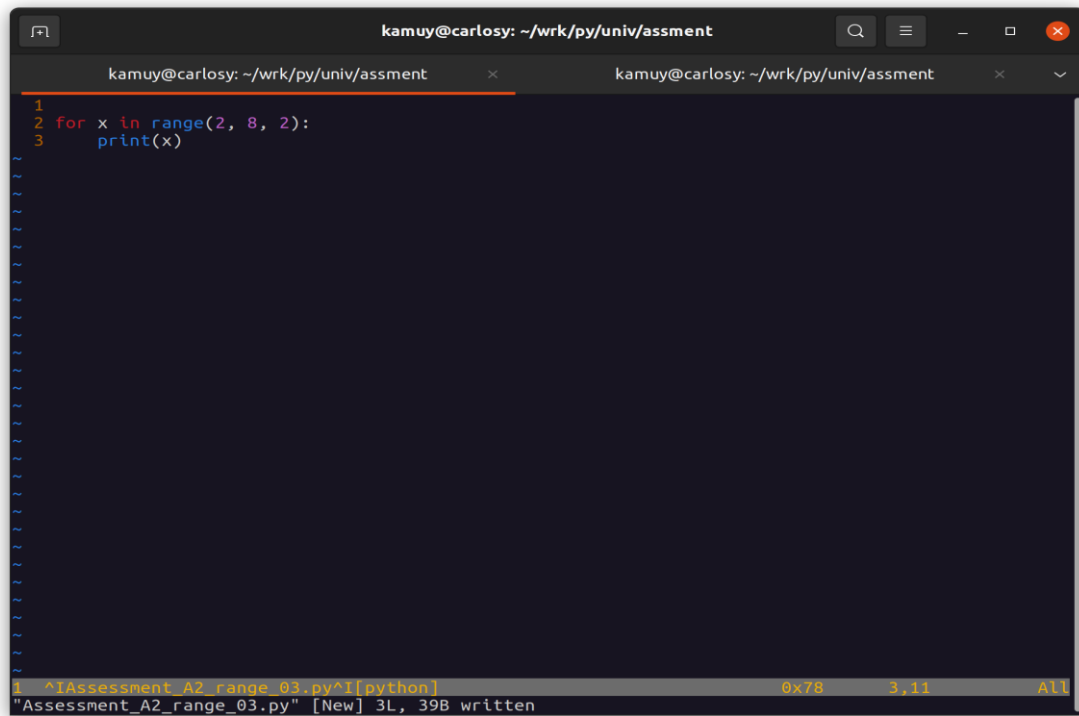
Output



```
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_02.py
2
3
4
kamuy@carlosy:~/wrk/py/univ/assment$
```

Python **range()** function with three parameters (start, stop and step):

Prints out 2, 4, 6

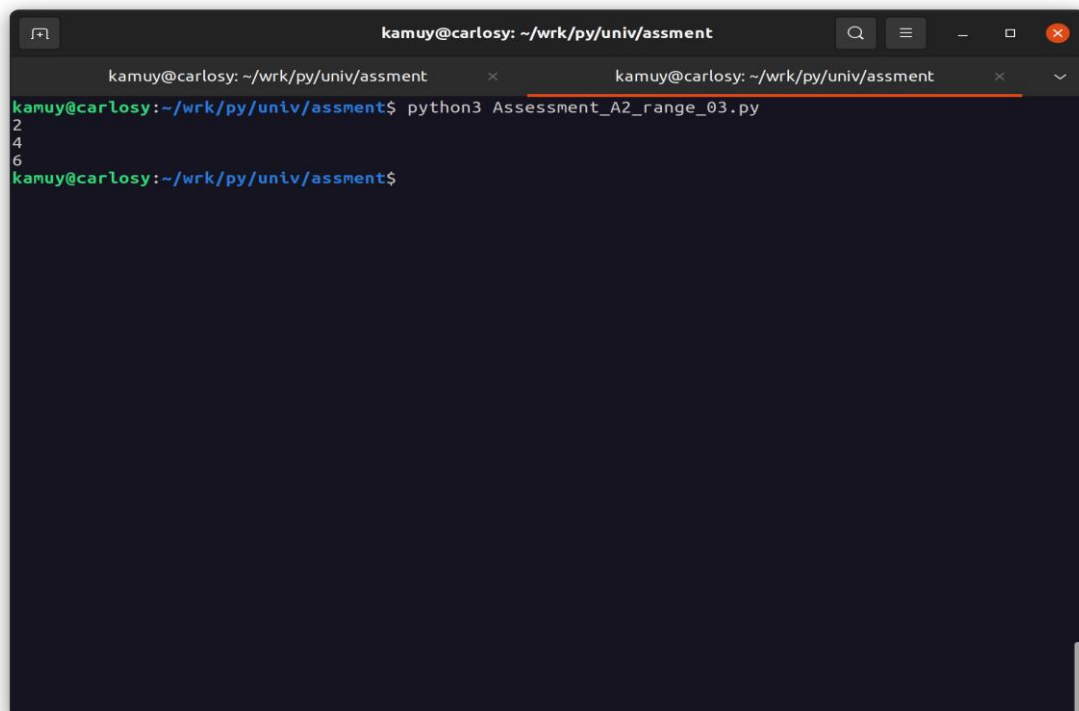


A screenshot of a code editor window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The editor contains a Python script with the following code:

```
1
2 for x in range(2, 8, 2):
3     print(x)
```

The status bar at the bottom indicates the file is "Assessment_A2_range_03.py" [New] 3L, 39B written.

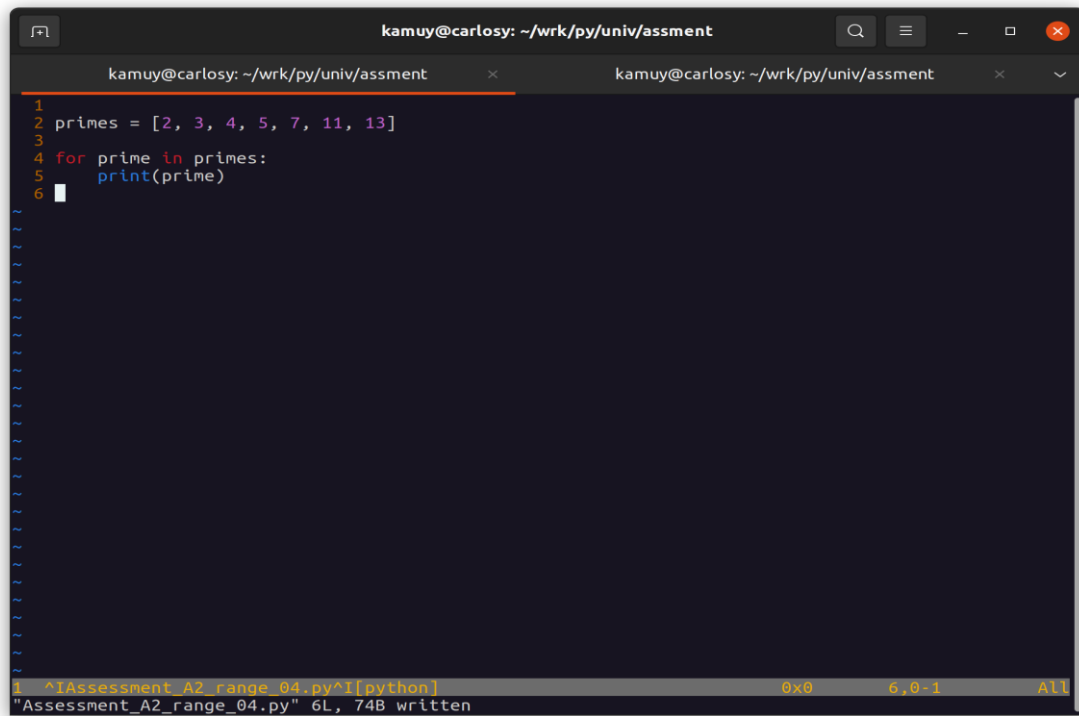
Output



A screenshot of a terminal window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The terminal shows the command "python3 Assessment_A2_range_03.py" being executed, resulting in the output:

```
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_03.py
2
4
6
kamuy@carlosy:~/wrk/py/univ/assment$
```

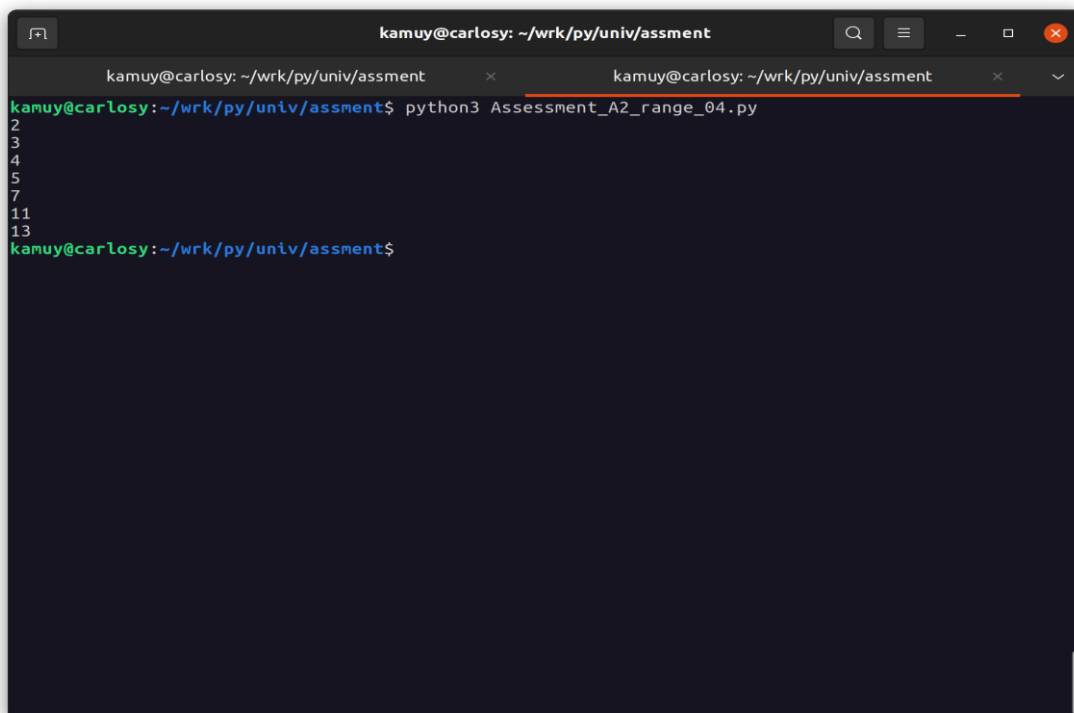
Loop Through a List



```
1 primes = [2, 3, 4, 5, 7, 11, 13]
2
3
4 for prime in primes:
5     print(prime)
6
```

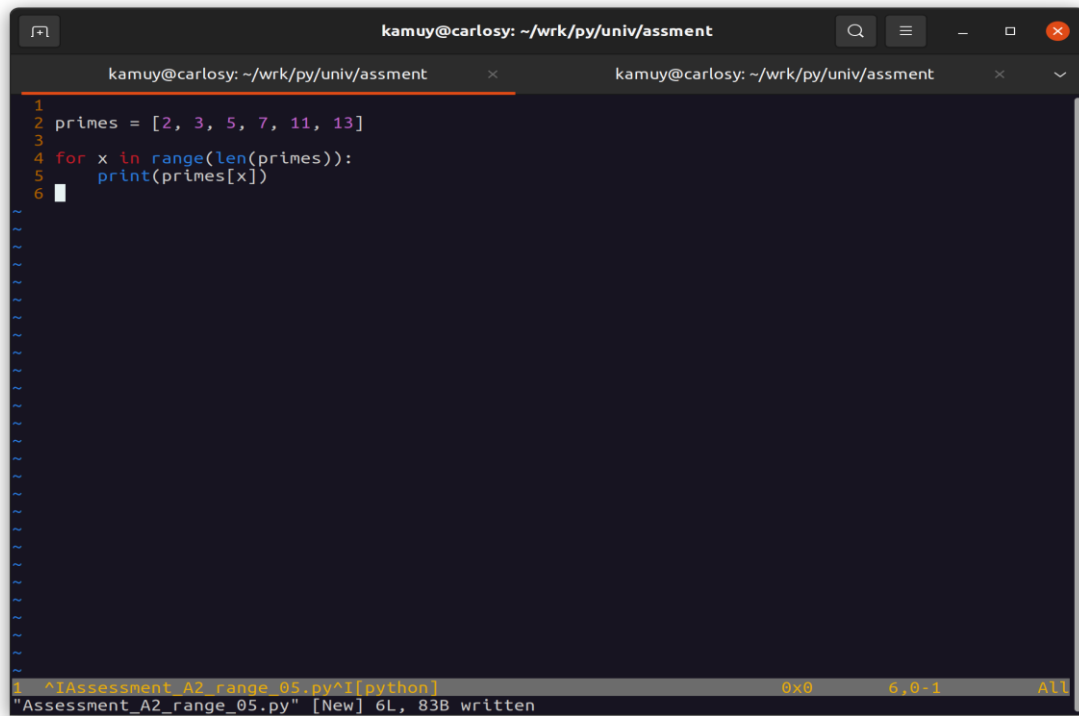
^IAssessment_A2_range_04.py^I[python] 0x0 6,0-1 All
"Assessment_A2_range_04.py" 6L, 74B written

Output



```
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_04.py
2
3
4
5
7
11
13
kamuy@carlosy:~/wrk/py/univ/assment$
```

In case we want to use the traditional for loop which print all items by referring to their index number

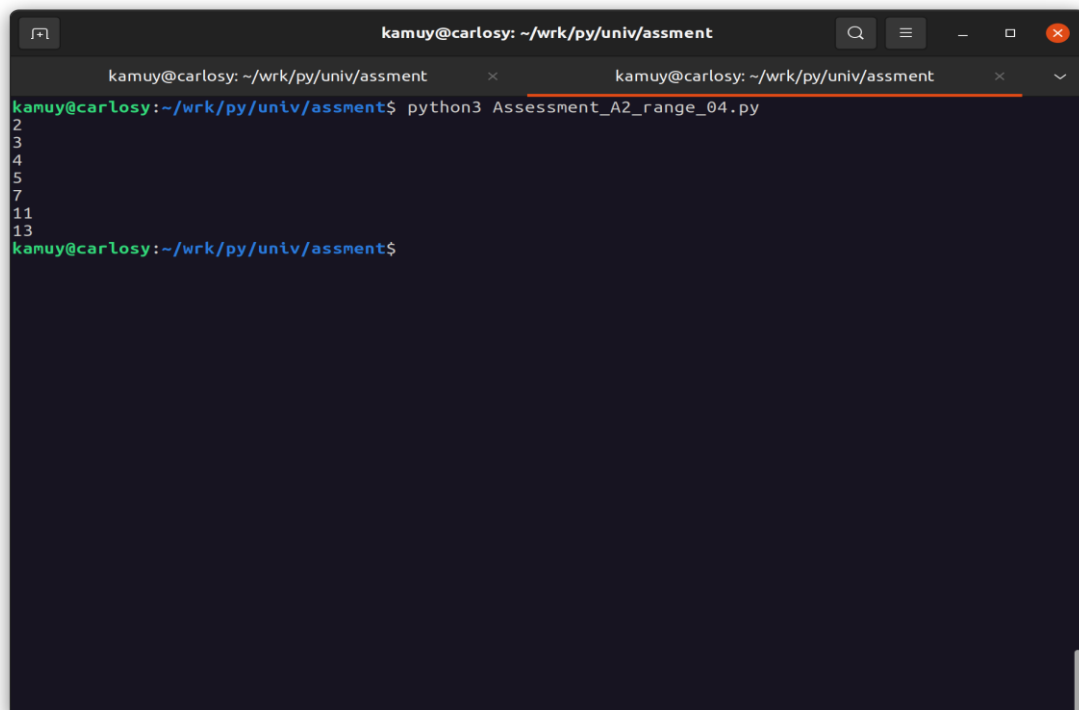


A screenshot of a code editor window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The editor contains a Python script with the following code:

```
1 primes = [2, 3, 5, 7, 11, 13]
2
3
4 for x in range(len(primes)):
5     print(primes[x])
6
```

The status bar at the bottom indicates the file is "Assessment_A2_range_05.py" [New] 6L, 83B written.

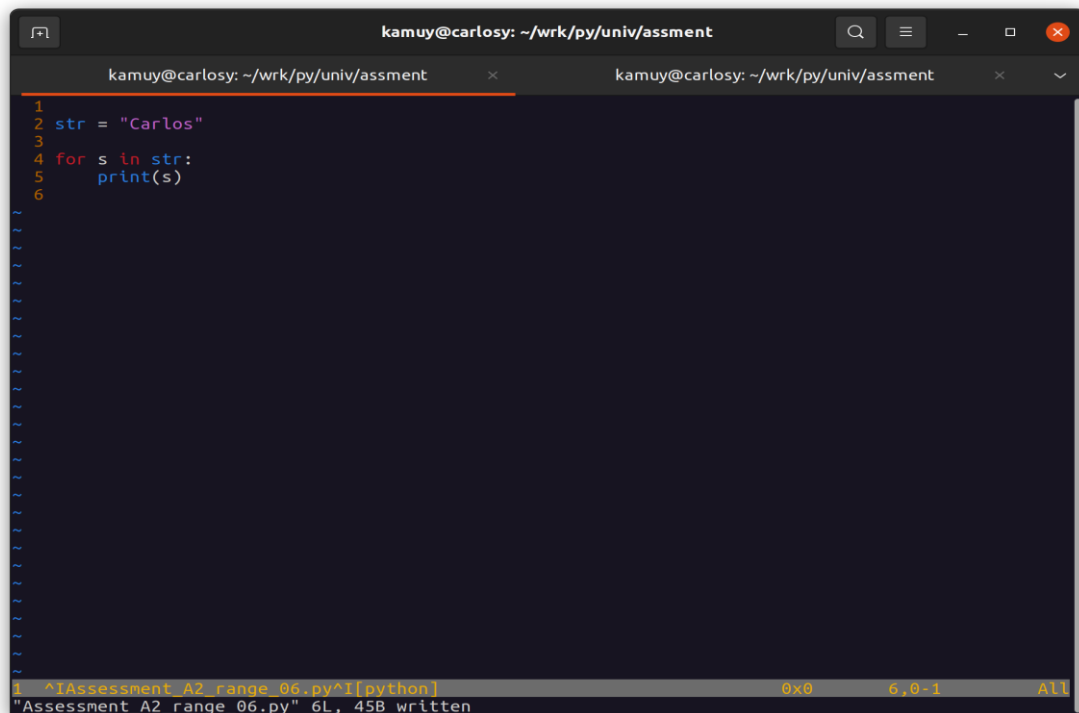
Output



A screenshot of a terminal window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The terminal shows the command "python3 Assessment_A2_range_04.py" being executed, followed by the output of the script:

```
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_04.py
2
3
4
5
7
11
13
kamuy@carlosy:~/wrk/py/univ/assment$
```

Using for loop in Python string

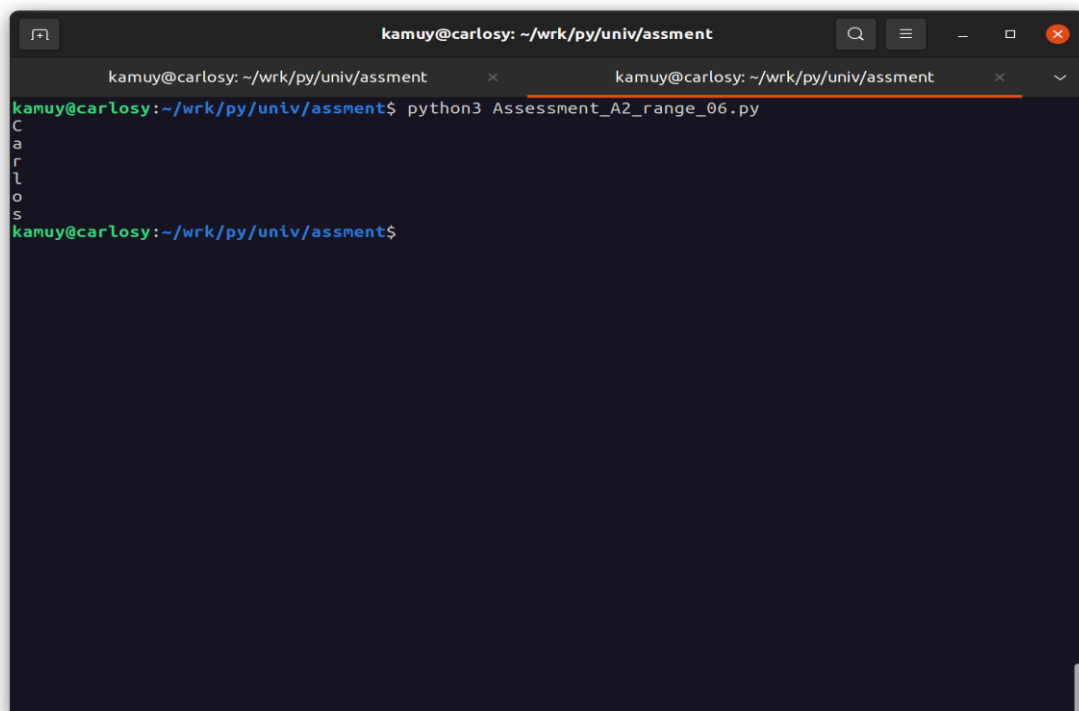


A screenshot of a code editor window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The editor contains a Python script with the following code:

```
1 str = "Carlos"
2
3
4 for s in str:
5     print(s)
6
```

The script is saved as "Assessment_A2_range_06.py" and is 6 lines long and 45 bytes in size. The status bar at the bottom indicates "0x0 6,0-1 All".

Output



A screenshot of a terminal window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The terminal shows the command "python3 Assessment_A2_range_06.py" being executed, and the output is the string "Carlos" printed character by character on separate lines:

```
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_06.py
C
a
r
l
o
s
kamuy@carlosy:~/wrk/py/univ/assment$
```

Nested for loop syntax in Python [4]

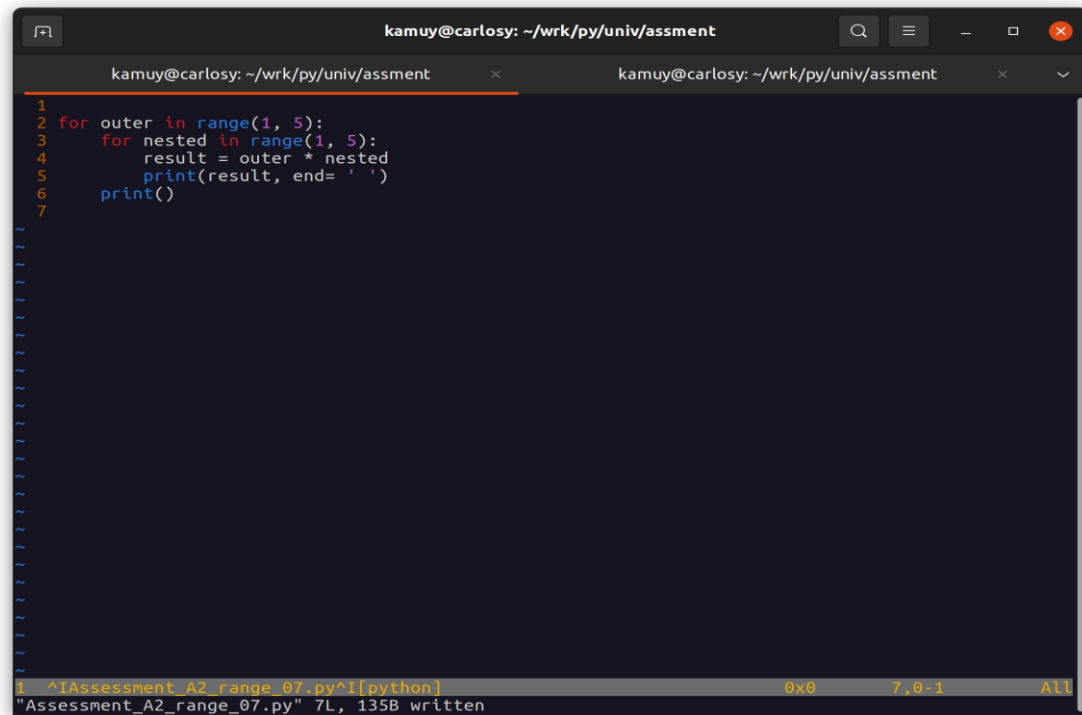
for iteration_var in sequence:

 for iteration_var in sequence:

 statements(s)

 statements(s)

For loop example

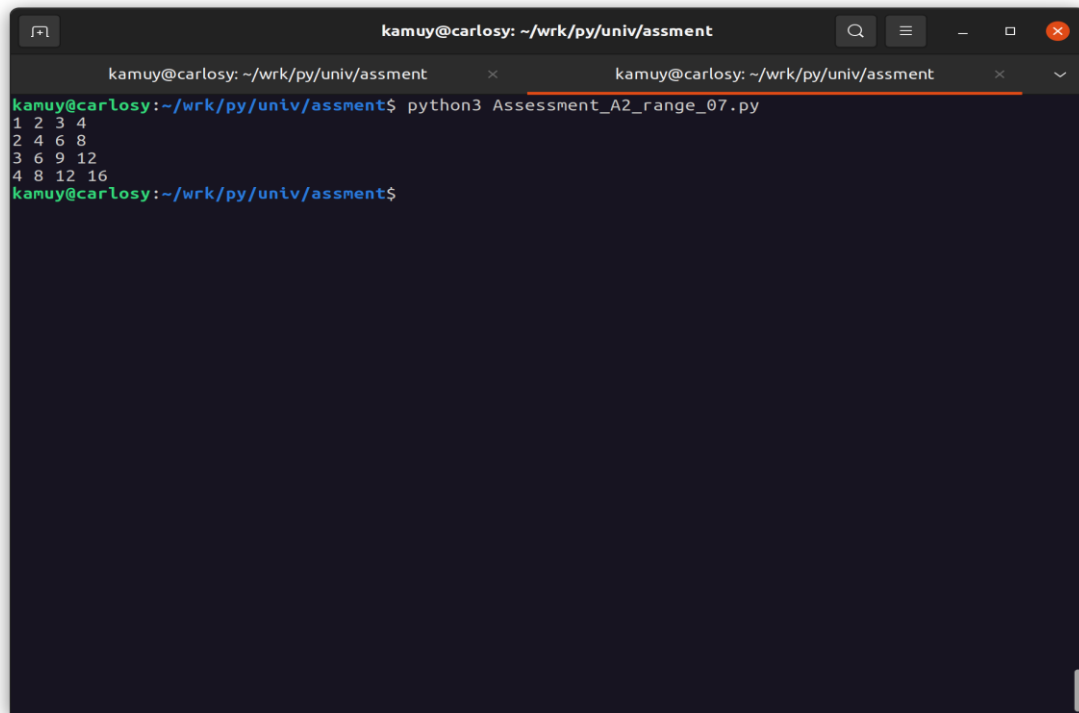


The screenshot shows a terminal window with a dark background. The title bar indicates the user is 'kamuy@carlosy' in the directory '~/wrk/py/univ/assment'. The terminal contains a Python script with a nested for loop. The script is as follows:

```
1
2 for outer in range(1, 5):
3     for nested in range(1, 5):
4         result = outer * nested
5         print(result, end= ' ')
6     print()
7
```

Below the script, there are several tilde (~) characters representing the output of the program. At the bottom of the terminal, a status bar shows the file path '^IAssessment_A2_range_07.py^I[prompt]', the cursor position '0x0', the line and column numbers '7,0-1', and the text 'All'. A message at the very bottom states '"Assessment_A2_range_07.py" 7L, 135B written'.

Output:

A terminal window with a dark background. The title bar shows 'kamuy@carlosoy: ~/wrk/py/univ/assment'. There are two tabs open, both with the same title. The terminal shows the command 'python3 Assessment_A2_range_07.py' being executed. The output is a 4x4 grid of numbers: 1 2 3 4, 2 4 6 8, 3 6 9 12, and 4 8 12 16. The prompt 'kamuy@carlosoy:~/wrk/py/univ/assment\$' is visible at the bottom.

```
kamuy@carlosoy: ~/wrk/py/univ/assment
kamuy@carlosoy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_07.py
1 2 3 4
2 4 6 8
3 6 9 12
4 8 12 16
kamuy@carlosoy:~/wrk/py/univ/assment$
```

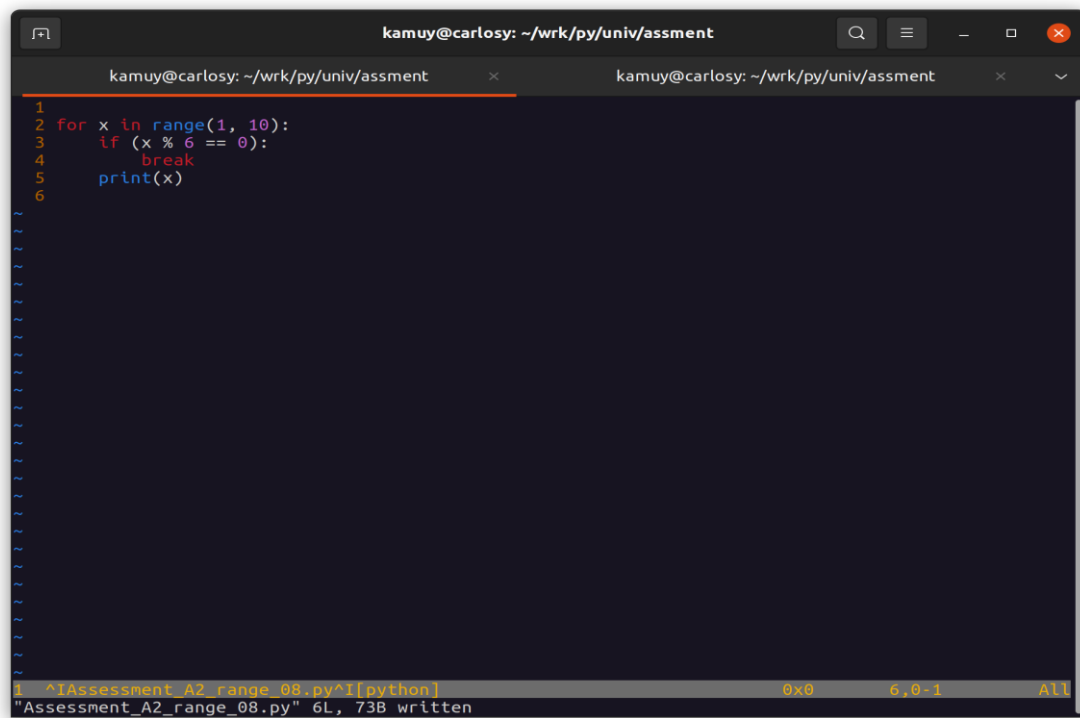
In this code the program first encounters the outer loop, executing its first iteration. The **first iteration** triggers the nested loop, which then runs the code to completion. Then the program returns back to the outer loop, completing the **second iteration** and triggering the nested loop. Then, the nested loop runs to completion, and the program returns back to the top of the outer loop until the sequence is complete or a break or continue statement disrupts the process. The print() function in inner loop has second parameter end=' ' which appends a space instead of default newline. Therefore, the numbers will appear in one row.

"break" and "continue" statements

break is used to exit a for loop or a while loop, whereas **continue** is used to skip the current block, and return to the "for" or "while" statement.

An example using break statement.

Prints out – 1, 2, 3, 4, 5

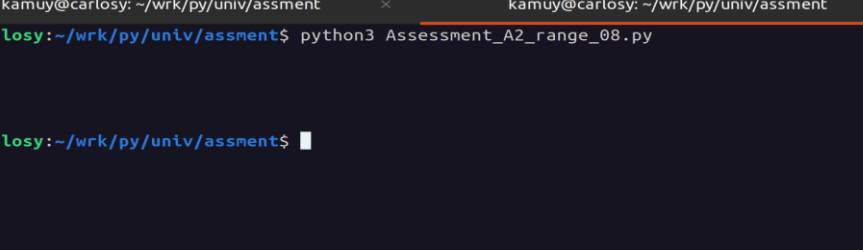


The image shows a code editor window with a dark theme. The title bar at the top reads "kamuy@carlosy: ~/wrk/py/univ/assment". There are two tabs open, both with the same title. The active tab contains a Python script with the following code:

```
1
2 for x in range(1, 10):
3     if (x % 6 == 0):
4         break
5     print(x)
6
```

Below the code, there are several lines of tilde (~) characters, indicating that the code is being executed or that there is a large amount of output. At the bottom of the editor, a status bar shows the file name "Assessment_A2_range_08.py" and the text "6L, 73B written".

Output

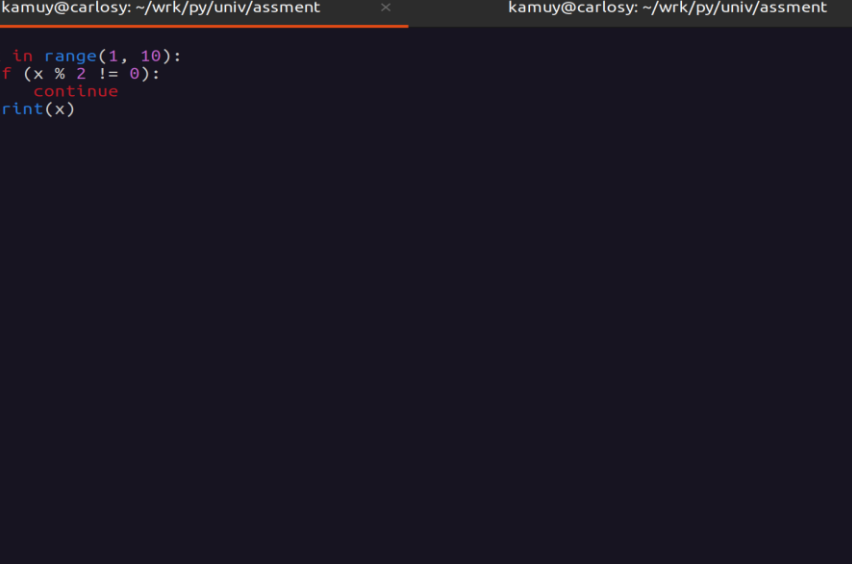


The screenshot shows a terminal window with a dark background. At the top, there is a title bar with the text "kamuy@carlosoy: ~/wrk/py/univ/assment" and standard window control buttons (minimize, maximize, close). Below the title bar, there are two tabs, both labeled "kamuy@carlosoy: ~/wrk/py/univ/assment". The terminal content shows a prompt "kamuy@carlosoy:~/wrk/py/univ/assment\$" followed by the command "python3 Assessment_A2_range_08.py". The command is executed, and the prompt returns. There are some faint, illegible characters at the bottom of the terminal window.

```
kamuy@carlosoy: ~/wrk/py/univ/assment
kamuy@carlosoy: ~/wrk/py/univ/assment
kamuy@carlosoy:~/wrk/py/univ/assment$ python3 Assessment_A2_range_08.py
kamuy@carlosoy:~/wrk/py/univ/assment$
```

An **example** using continue statement.

Prints out only even numbers - 2, 4, 6, 8



The screenshot shows a terminal window with a dark background. At the top, there are two tabs, both labeled "kamuy@carlosoy: ~/wrk/py/univ/assment". The first tab is active. The terminal displays a Python script with the following code:

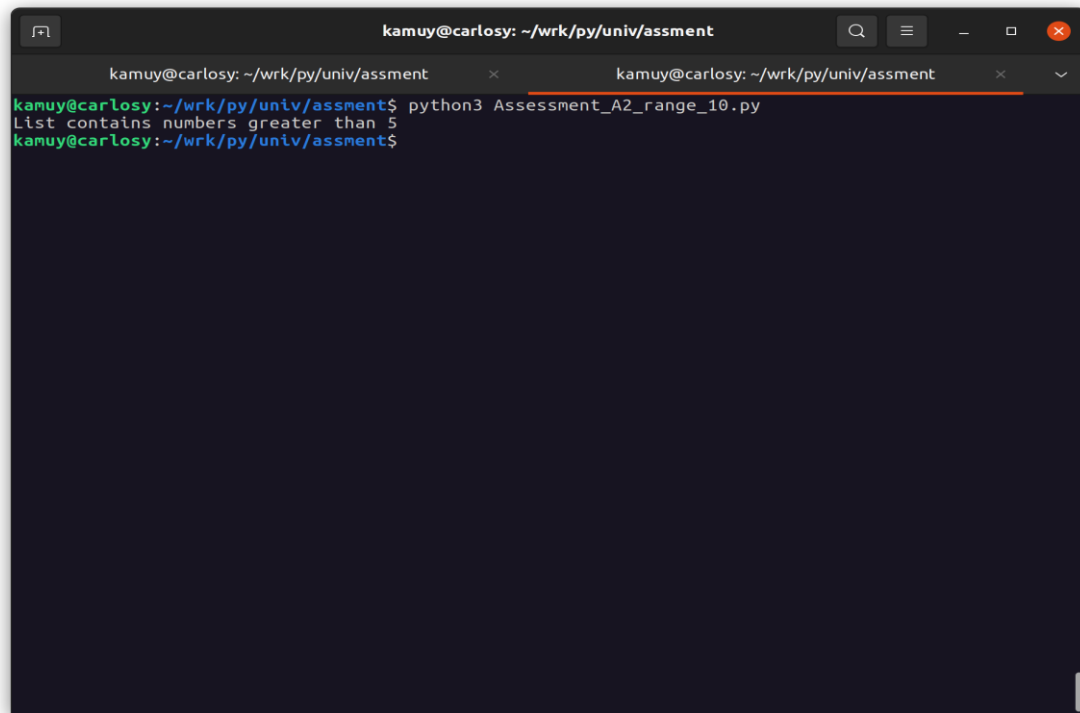
```
1
2 for x in range(1, 10):
3     if (x % 2 != 0):
4         continue
5     print(x)
6
```

The script is executed, and the output is displayed in the terminal:

```
1 ^IAssessment_A2_range_09.py^I[python]
2 "Assessment A2 range 09.py" 6L, 76B written
```

The output shows the execution of the script, with the file name and size of the output file displayed.

Output

A terminal window with a dark background. The title bar shows 'kamuy@carlosoy: ~/wrk/py/univ/assment'. There are two tabs open, both with the same title. The active tab shows the command 'python3 Assessment_A2_range_10.py' being executed. The output of the script is 'List contains numbers greater than 5'. The prompt 'kamuy@carlosoy: ~/wrk/py/univ/assment\$' is visible at the bottom.

```
kamuy@carlosoy: ~/wrk/py/univ/assment
kamuy@carlosoy: ~/wrk/py/univ/assment$ python3 Assessment_A2_range_10.py
List contains numbers greater than 5
kamuy@carlosoy: ~/wrk/py/univ/assment$
```

A3.

A **list** is data type, which can hold **multiple** objects. One important aspect of list structure is that list is mutable, and can be changed after it was created and also allows to add and remove objects from the sequence. A **variable** is a "kind-of" storage location with a name attached to it, and it can contain any **single** object.

Lists are very similar to arrays. They can contain any type of variable, and they can contain as many variables. Lists can also be iterated over and list literals are written within square brackets [] with the first element at index 0. There are many methods associated to them.

Creating list

Creating a list is as simple as putting different comma-separated values in square brackets.

```
a_list = [1, 2, 3, 4, 5]
```

```
b_list = ['a', 'b', 'c', 'd', 'e']
```

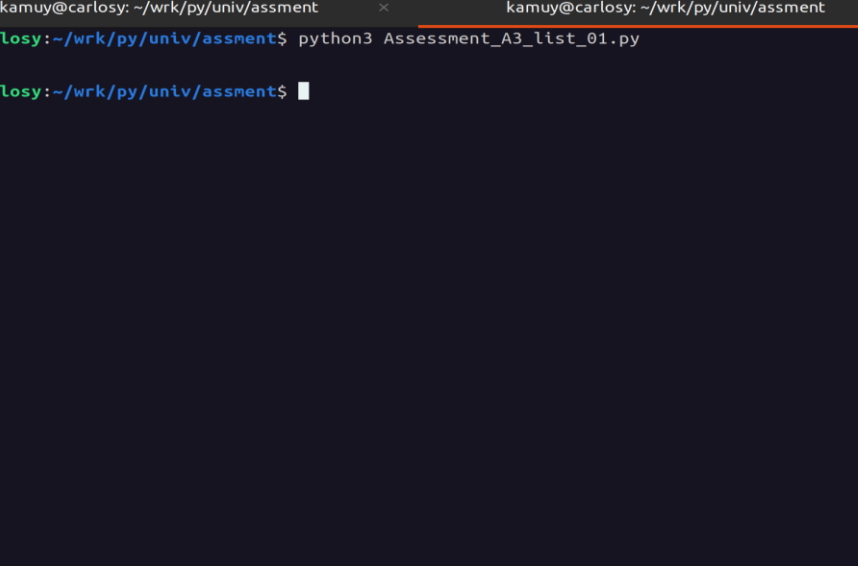
```
c_list = ['one', 'two', 'three', 'four', 'five']
```

```
d_list = [1, 2, 'three', 'four', 'five']
```

Accessing list values

Example

Output



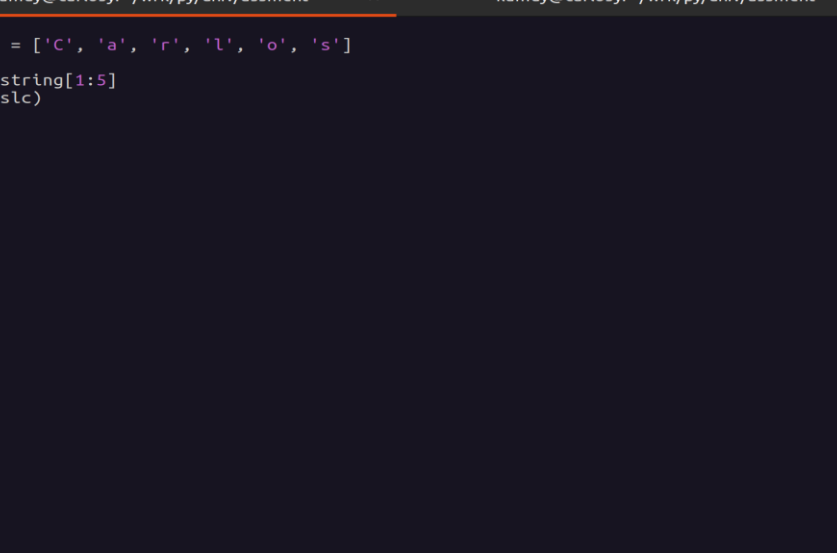
The screenshot shows a terminal window with a dark background. At the top, there are two tabs, both labeled "kamuy@carlosy: ~/wrk/py/univ/assment". The active tab is highlighted with an orange underline. The terminal content shows a prompt "kamuy@carlosy:~/wrk/py/univ/assment\$" followed by the command "python3 Assessment_A3_list_01.py". The output of the script is displayed on the next two lines: "2" and "three". A new prompt "kamuy@carlosy:~/wrk/py/univ/assment\$" is visible on the third line, with a white cursor character.

```
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A3_list_01.py
2
three
kamuy@carlosy:~/wrk/py/univ/assment$
```

Slicing elements of a list

Python slice extracts elements, based on a start and stop.

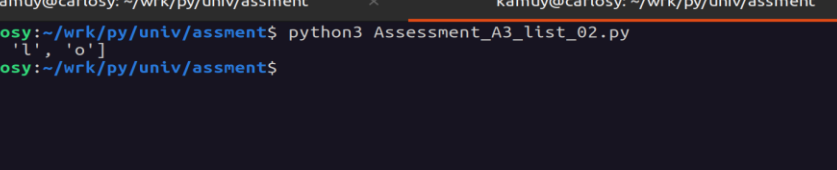
string[1:5] - The 1 means to start at second element in the list (the slicing index starts at 0). The 5 means to end at the sixth element in the list, but not include it. The colon in the middle is how Python's lists recognise that we want to use slicing to get elements in the list.



```
kamuy@carloso: ~/wrk/py/univ/assment
kamuy@carloso: ~/wrk/py/univ/assment
1
2 string = ['C', 'a', 'r', 'l', 'o', 's']
3
4 slc = string[1:5]
5 print(slc)
6

^IAssessment_A3_list_02.py^I[python]
"Assessment A3 list 02.py" 6L, 72B written
```

Output

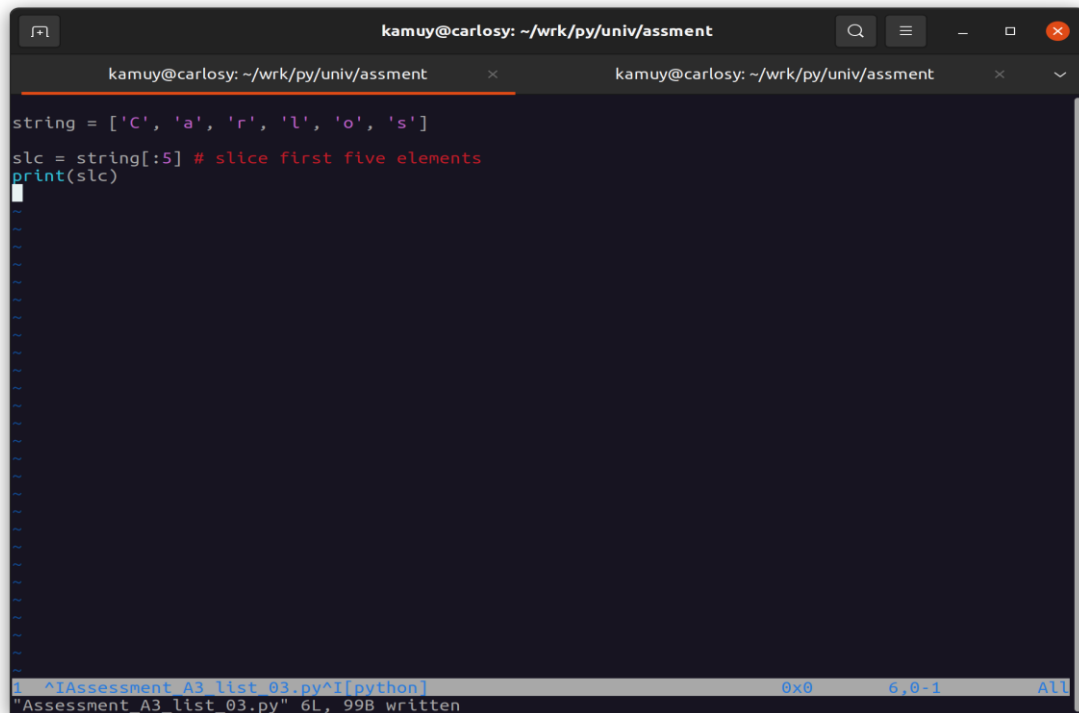


The screenshot shows a terminal window with a dark background. The title bar at the top reads "kamuy@carlosy: ~/wrk/py/univ/assment". Below the title bar, there are two tabs, both labeled "kamuy@carlosy: ~/wrk/py/univ/assment". The first tab is active. The terminal content shows the prompt "kamuy@carlosy:~/wrk/py/univ/assment\$" followed by the command "python3 Assessment_A3_list_02.py". The output of the command is displayed on the next line: "['a', 'r', 'l', 'o']".

```
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A3_list_02.py
['a', 'r', 'l', 'o']
kamuy@carlosy:~/wrk/py/univ/assment$
```

Example

`string[:5]` – Slice the first five elements starting at first element (index 0 to index 4)

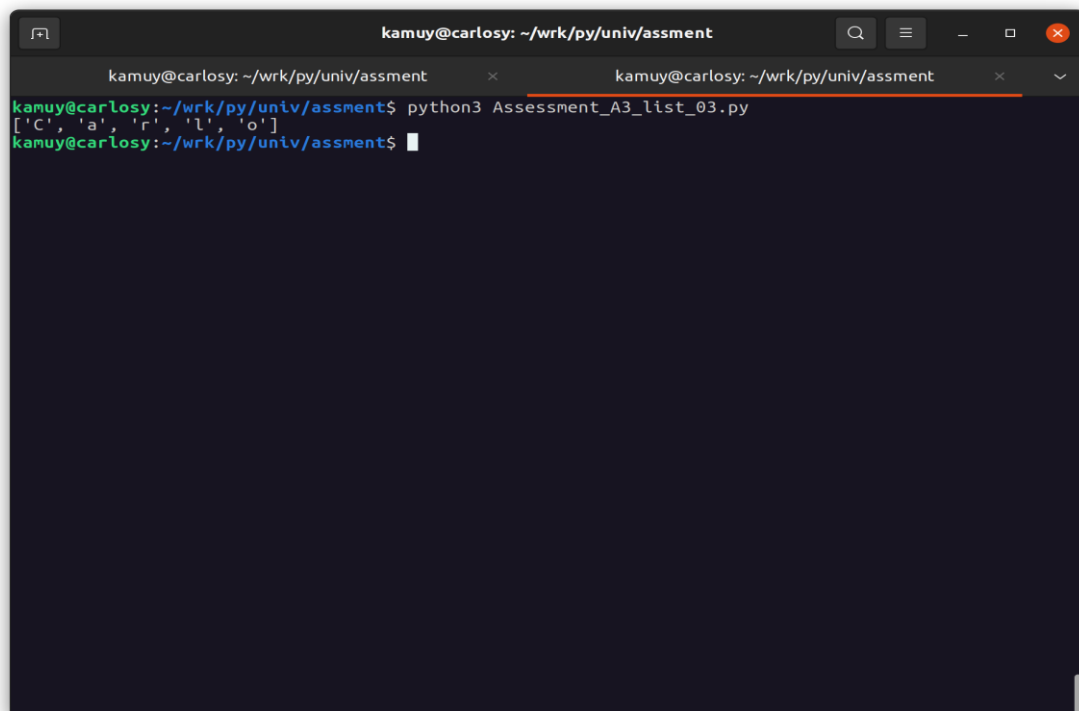


The screenshot shows a code editor window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The editor contains a Python script with the following code:

```
string = ['C', 'a', 'r', 'l', 'o', 's']  
slc = string[:5] # slice first five elements  
print(slc)
```

The script is saved as "Assessment_A3_list_03.py". The bottom status bar indicates the file size is 6L, 99B and the encoding is UTF-8.

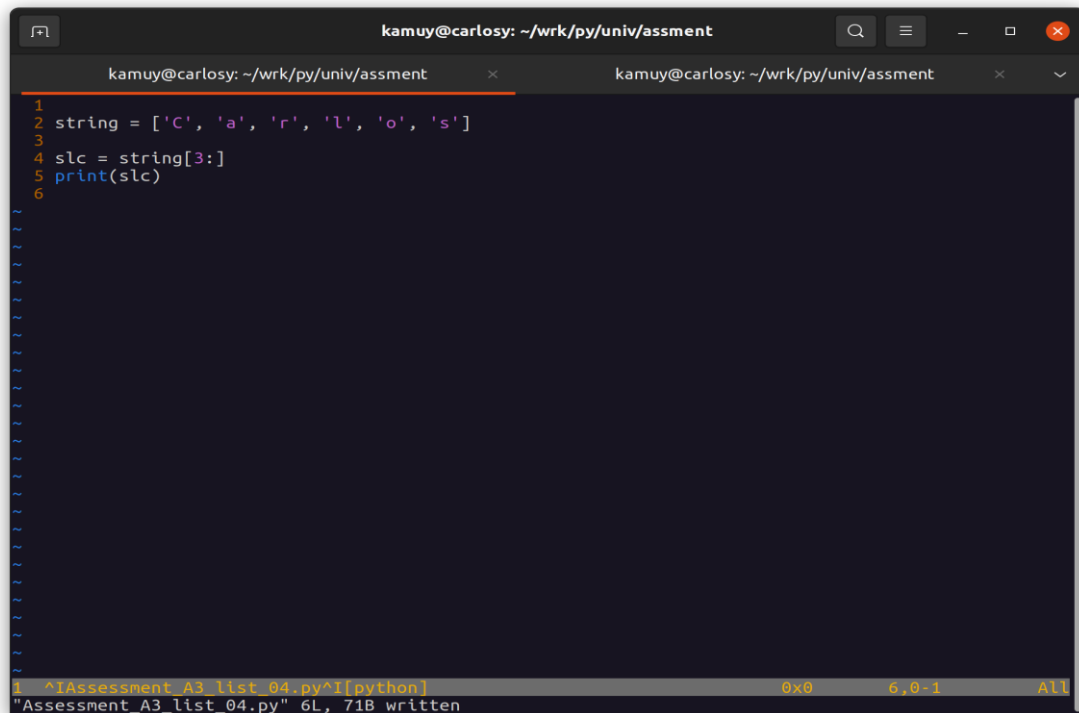
Output



The screenshot shows a terminal window titled "kamuy@carlosy: ~/wrk/py/univ/assment". The terminal displays the command to run the script and its output:

```
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A3_list_03.py  
['C', 'a', 'r', 'l', 'o']  
kamuy@carlosy:~/wrk/py/univ/assment$
```

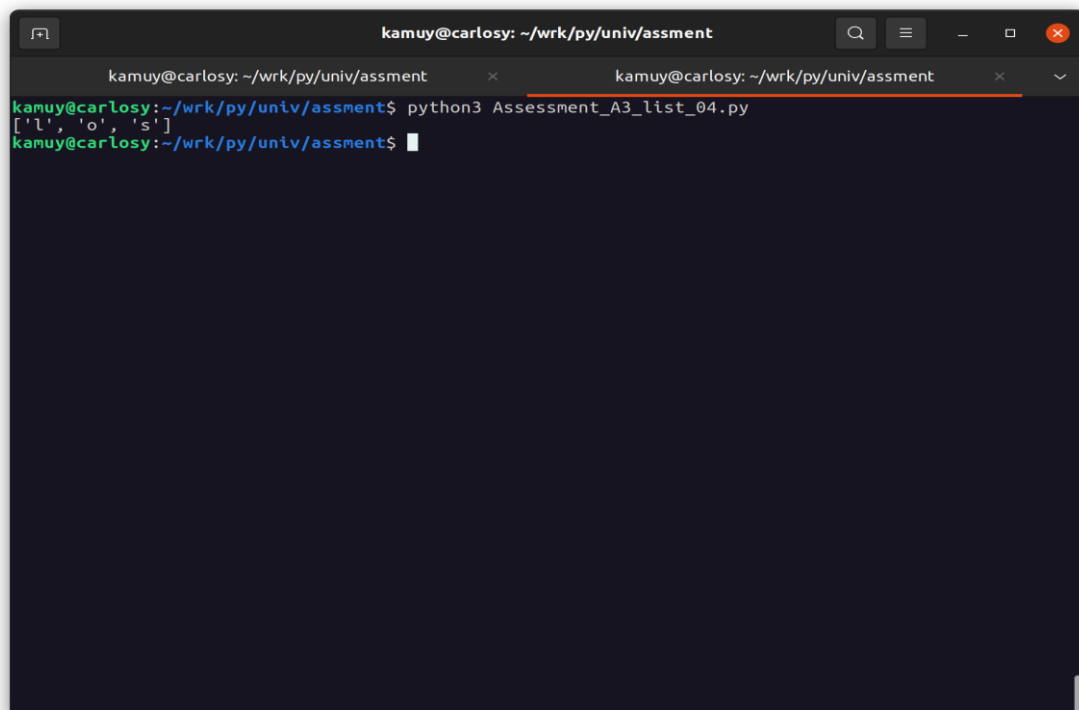
Slice from 4th element, Python starts its lists at 0 rather than 1.



```
1 string = ['C', 'a', 'r', 'l', 'o', 's']
2
3
4 slc = string[3:]
5 print(slc)
6
```

^IAssessment_A3_list_04.py^I[python] 0x0 6,0-1 All
"Assessment_A3_list_04.py" 6L, 71B written

Output



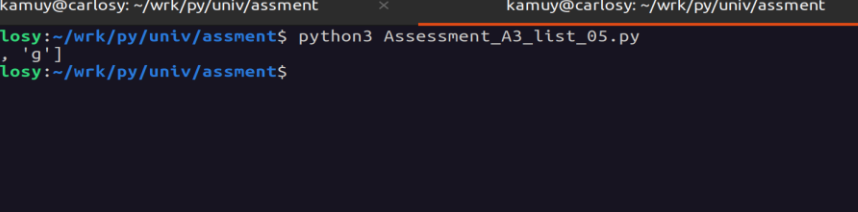
```
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy:~/wrk/py/univ/assment$ python3 Assessment_A3_list_04.py
['l', 'o', 's']
kamuy@carlosy:~/wrk/py/univ/assment$
```

Slice with Negative Indices

It's possible to specify negative indices while slicing a list

[illegible]

Output

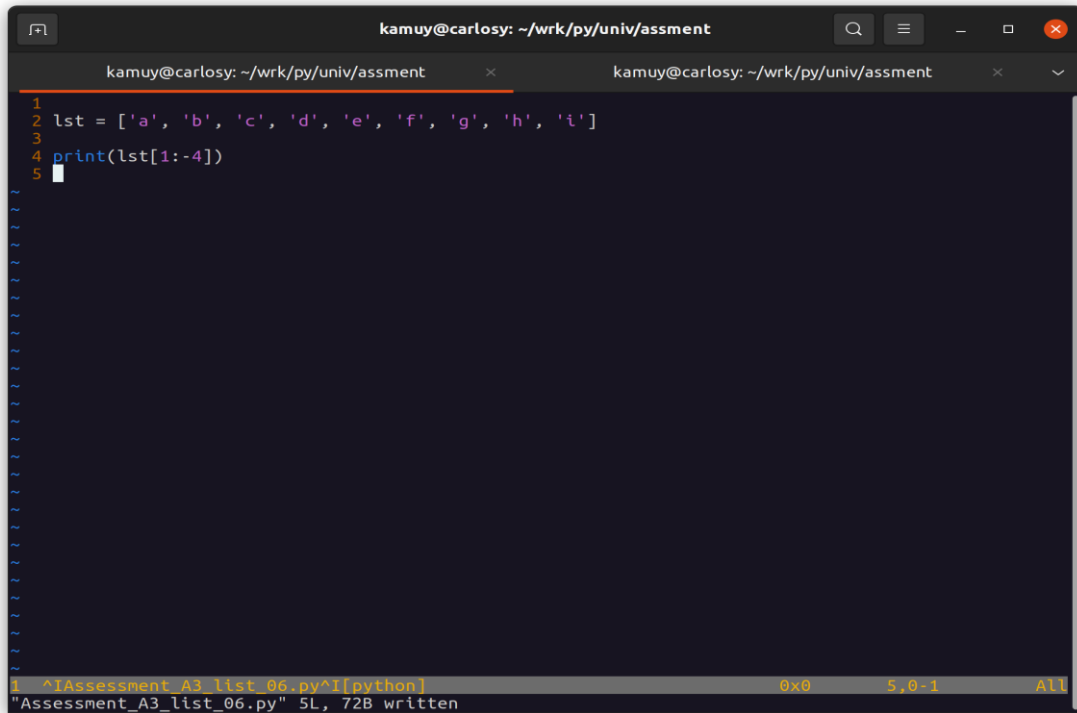


The screenshot shows a terminal window with a dark background. At the top, there are two tabs, both labeled "kamuy@carlosoy: ~/wrk/py/univ/assment". The first tab is active. The terminal content shows the command "python3 Assessment_A3_list_05.py" being executed, which returns the output "['e', 'f', 'g']". The prompt "kamuy@carlosoy:~/wrk/py/univ/assment\$" is visible on both lines.

```
kamuy@carlosoy: ~/wrk/py/univ/assment
kamuy@carlosoy:~/wrk/py/univ/assment$ python3 Assessment_A3_list_05.py
['e', 'f', 'g']
kamuy@carlosoy:~/wrk/py/univ/assment$
```

Slice with Positive & Negative Indices

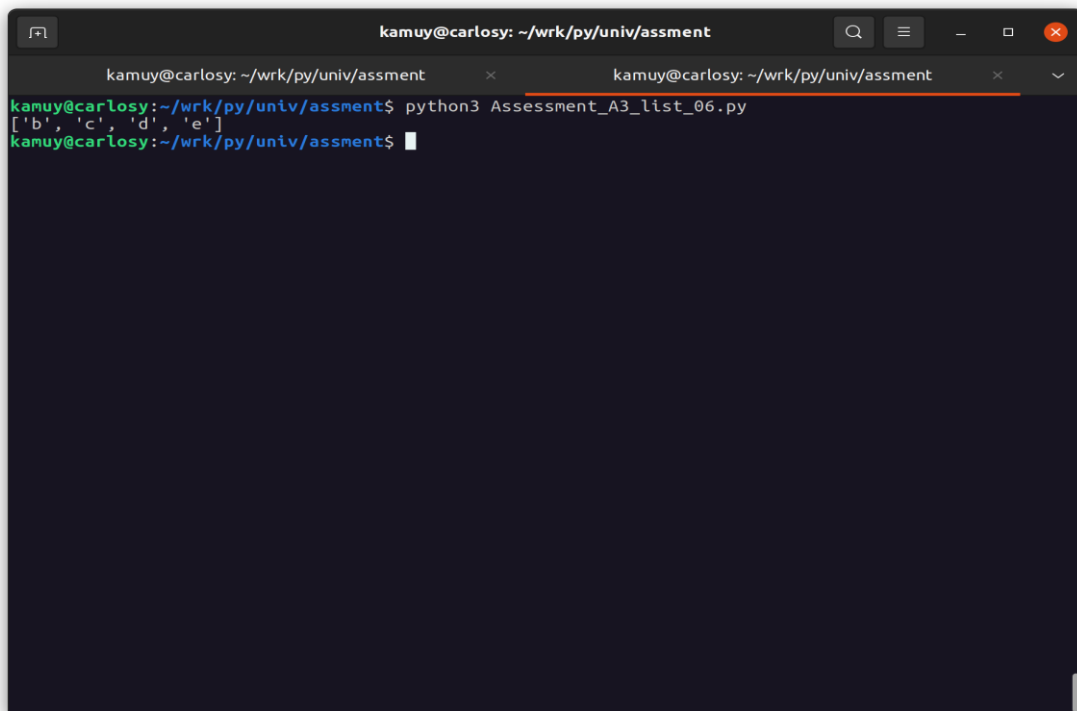
As well, it is possible to specify both positive and negative indices at the same time.



```
1  
2 lst = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']  
3  
4 print(lst[1:-4])  
5
```

The screenshot shows a code editor window with the title "kamuy@carlosoy: ~/wrk/py/univ/assment". The editor contains a Python script with five lines. Line 1 is empty. Line 2 defines a list 'lst' with elements 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'. Line 3 is empty. Line 4 prints the slice 'lst[1:-4]'. Line 5 is empty. The status bar at the bottom indicates the file is "Assessment_A3_list_06.py" and has 5 lines and 72 bytes.

Output



```
kamuy@carlosoy:~/wrk/py/univ/assment$ python3 Assessment_A3_list_06.py  
['b', 'c', 'd', 'e']  
kamuy@carlosoy:~/wrk/py/univ/assment$
```

The screenshot shows a terminal window with the title "kamuy@carlosoy: ~/wrk/py/univ/assment". The terminal shows the command "python3 Assessment_A3_list_06.py" being executed, which outputs the list ['b', 'c', 'd', 'e']. The prompt "kamuy@carlosoy:~/wrk/py/univ/assment\$" is shown again on the next line.

Part B

Exercise 1

```
total = 0
```

```
name = input("Customer name: ")
```

```
price = float(input("What is the price of product? "))
```

```
quantity = int(input("How much is the quantity? "))
```

```
print()
```

```
if (quantity < 0 or price < 0):
```

```
print("Quantity or price can't be negative.")
```

```
else:
```

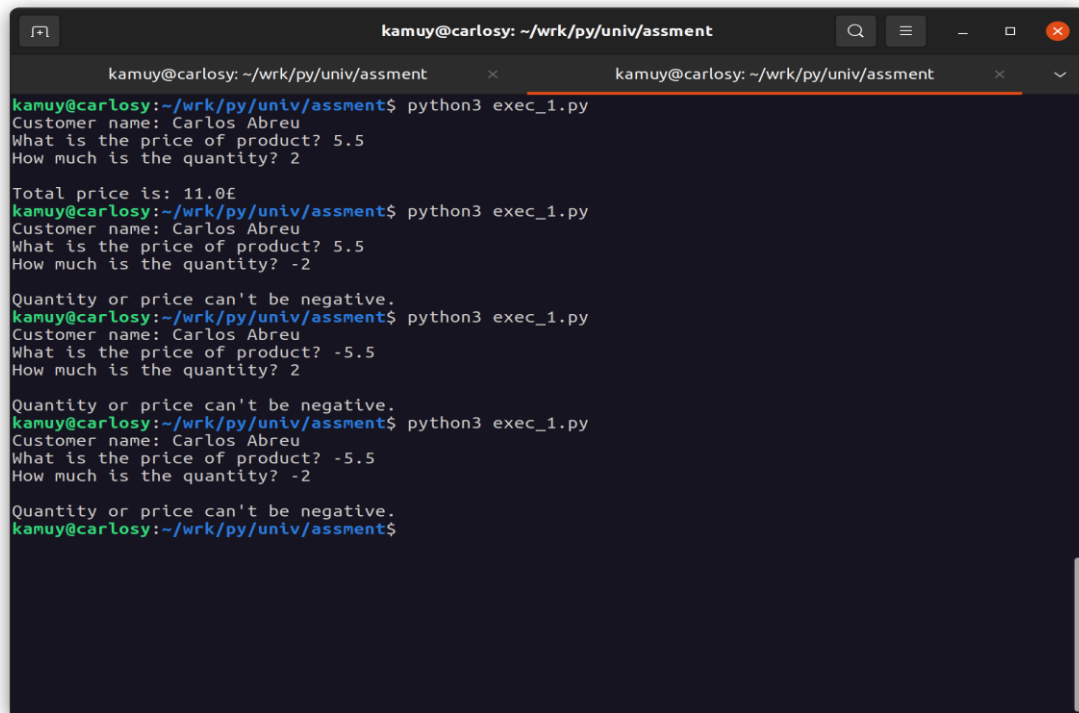
```
total = price * quantity
```

```
print(f"Total price is: {total}£")
```

(Code)

```
1
2 total = 0
3
4 name = input("Customer name: ")
5 price = float(input("What is the price of product? "))
6 quantity = int(input("How much is the quantity? "))
7
8 print()
9
10 if (quantity < 0 or price < 0):
11     print("Quantity or price can't be negative.")
12 else:
13     total = price * quantity
14     print(f"Total price is: {total}€")
15
```

Output

A terminal window titled 'kamuy@carlosy: ~/wrk/py/univ/assment' showing the execution of a Python script 'exec_1.py'. The script prompts for customer name, price, and quantity. It calculates the total price as 11.0€ for valid inputs (5.5, 2). It also shows error handling for negative quantities or prices, displaying 'Quantity or price can't be negative.' for inputs like -2 or -5.5.

```
kamuy@carlosy: ~/wrk/py/univ/assment$ python3 exec_1.py
Customer name: Carlos Abreu
What is the price of product? 5.5
How much is the quantity? 2

Total price is: 11.0€
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_1.py
Customer name: Carlos Abreu
What is the price of product? 5.5
How much is the quantity? -2

Quantity or price can't be negative.
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_1.py
Customer name: Carlos Abreu
What is the price of product? -5.5
How much is the quantity? 2

Quantity or price can't be negative.
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_1.py
Customer name: Carlos Abreu
What is the price of product? -5.5
How much is the quantity? -2

Quantity or price can't be negative.
kamuy@carlosy:~/wrk/py/univ/assment$
```

Exercise 2 (code)

Note: Because the source code for this exercise doesn't fit into one screen I decided to paste the entire code here, and show only the image of the code running.

```
import math
```

```
radius_inner= float(input("Enter the inner radius: "))
```

```
radius_outer = float(input("Enter the outer radius: "))
```

```
enter_area = input("Do you want to calculate the shaded area? (y/Y/Yes) or (n/N//No): ")
```

```
list_area = ['y', 'Y', 'Yes', 'yes', 'YES']
```

```
while (enter_area in list_area):
```

```
    if (radius_inner < radius_outer):
```

```
        area_inner = math.pi * radius_inner * radius_inner
```

```
        area_outer = math.pi * radius_outer * radius_outer
```

```
        area_shaded = area_outer - area_inner
```

```
print("The shaded area is: {0:.2f}".format(area_shaded))  
break
```

```
elif (radius_inner >= radius_outer):
```

```
    print("Inner area must be smaller than outer area!")
```

```
    enter_area = input("Do you want to try again? (y/Y/Yes) or (n/N/No) ")
```

```
    list_area = ['n', 'N', 'No']
```

```
    if (enter_area in list_area):
```

```
        break
```

```
radius_inner= float(input("Enter the inner radius: "))
```

```
radius_outer = float(input("Enter the outer radius: "))
```

```
list_area = ['y', 'Y', 'Yes', 'yes', 'YES']
```

```
if (radius_inner < radius_outer):
```

```
    area_inner = math.pi * radius_inner * radius_inner
```

```
    area_outer = math.pi * radius_outer * radius_outer
```

```
    area_shaded = area_outer - area_inner
```

```
    print("The shaded area is: {0:.2f}".format(area_shaded))
```

```
    break
```

```
else:
```

```
    enter_area = input("Do you want to try again? (y/Y/Yes) or (n/N/No) ")
```

```
list_area = ['y', 'Y', 'Yes', 'yes', 'YES']
```

```
list_area_n = ['n', 'N', 'No', 'no', 'NO']
```

```
if enter_area in list_area:
```

```
    radius_inner= float(input("Enter the inner radius: "))
```

```
    radius_outer = float(input("Enter the outer radius: "))
```

```
if (radius_inner < radius_outer):  
    area_inner = math.pi * radius_inner * radius_inner  
    area_outer = math.pi * radius_outer * radius_outer  
    area_shaded = area_outer - area_inner  
    print("The shaded area is: {0:.2f}".format(area_shaded))
```

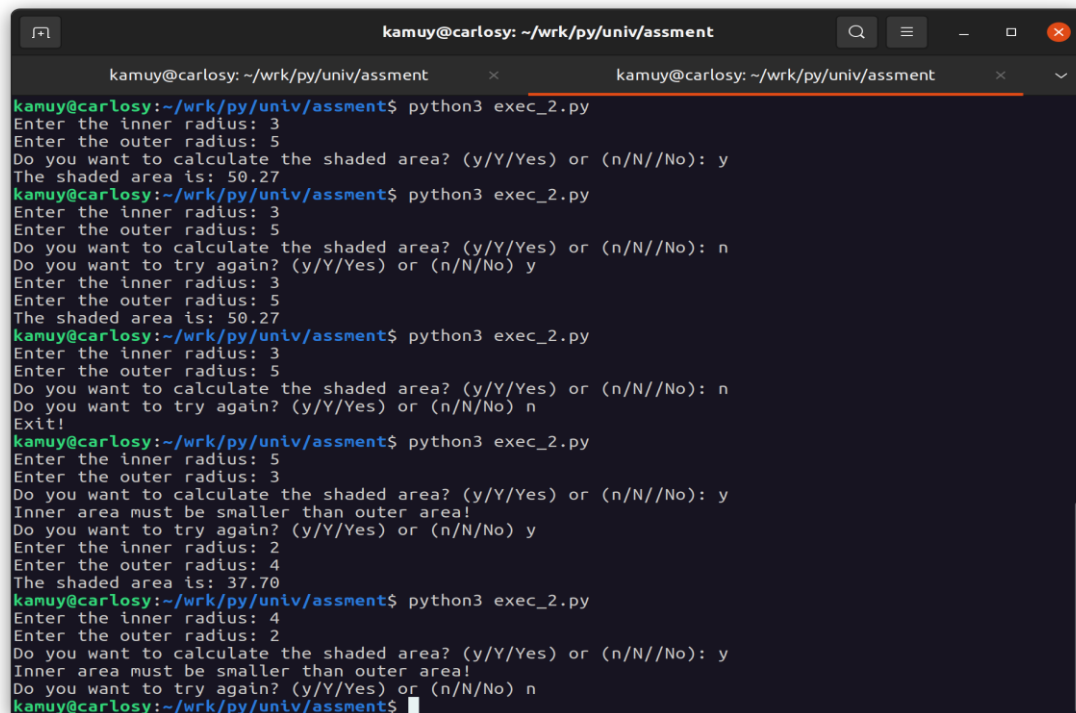
```
elif (radius_inner >= radius_outer):  
    print("Inner area must be smaller than outer area!")  
    enter_area = input("Do you want to try again? (y/Y/Yes) or (n/N/No) ")  
    list_area = ['y', 'Y', 'Yes', 'yes', 'YES']
```

```
if enter_area in list_area:  
    radius_inner= float(input("Enter the inner radius: "))  
    radius_outer = float(input("Enter the outter radius: "))
```

```
if (radius_inner < radius_outer):  
    area_inner = math.pi * radius_inner * radius_inner  
    area_outer = math.pi * radius_outer * radius_outer  
    area_shaded = area_outer - area_inner  
    print("The shaded area is: {0:.2f}".format(area_shaded))
```

```
else:  
    print("Exit!")
```

Output



```
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_2.py
Enter the inner radius: 3
Enter the outer radius: 5
Do you want to calculate the shaded area? (y/Y/Yes) or (n/N//No): y
The shaded area is: 50.27
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_2.py
Enter the inner radius: 3
Enter the outer radius: 5
Do you want to calculate the shaded area? (y/Y/Yes) or (n/N//No): n
Do you want to try again? (y/Y/Yes) or (n/N/No) y
Enter the inner radius: 3
Enter the outer radius: 5
The shaded area is: 50.27
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_2.py
Enter the inner radius: 3
Enter the outer radius: 5
Do you want to calculate the shaded area? (y/Y/Yes) or (n/N//No): n
Do you want to try again? (y/Y/Yes) or (n/N/No) n
Exit!
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_2.py
Enter the inner radius: 5
Enter the outer radius: 3
Do you want to calculate the shaded area? (y/Y/Yes) or (n/N//No): y
Inner area must be smaller than outer area!
Do you want to try again? (y/Y/Yes) or (n/N/No) y
Enter the inner radius: 2
Enter the outer radius: 4
The shaded area is: 37.70
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_2.py
Enter the inner radius: 4
Enter the outer radius: 2
Do you want to calculate the shaded area? (y/Y/Yes) or (n/N//No): y
Inner area must be smaller than outer area!
Do you want to try again? (y/Y/Yes) or (n/N/No) n
kamuy@carlosy:~/wrk/py/univ/assment$
```

Exercise 3 (Code)

```
total = 0

total_increase = 0

staff = []

salary = []

for counter in range(0, 5):

    print("Enter staff name: ")

    staff.append(input())

    print("Enter salary: ")

    salary.append(input())

    total = total + float(salary[counter])

    total_increase = total * 1.10
```

```
print()

print("Staff details entered: ")

for counter in range(0, 5):

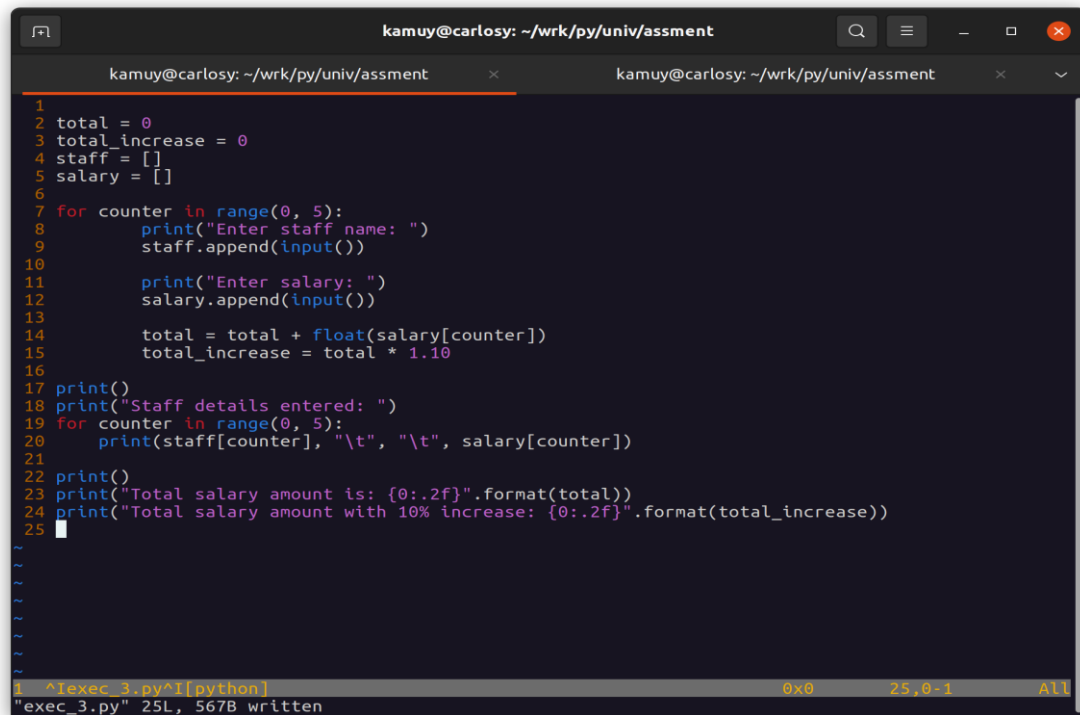
    print(staff[counter], "\t", "\t", salary[counter])

print()

print("Total salary amount is: {0:.2f}".format(total))

print("Total salary amount with 10% increase: {0:.2f}".format(total_increase))
```

(Code)



The screenshot shows a terminal window with a dark background and light-colored text. The window title is "kamuy@carlosy: ~/wrk/py/univ/assment". The code is as follows:

```
1 total = 0
2 total_increase = 0
3 staff = []
4 salary = []
5
6
7 for counter in range(0, 5):
8     print("Enter staff name: ")
9     staff.append(input())
10
11     print("Enter salary: ")
12     salary.append(input())
13
14     total = total + float(salary[counter])
15     total_increase = total * 1.10
16
17 print()
18 print("Staff details entered: ")
19 for counter in range(0, 5):
20     print(staff[counter], "\t", "\t", salary[counter])
21
22 print()
23 print("Total salary amount is: {0:.2f}".format(total))
24 print("Total salary amount with 10% increase: {0:.2f}".format(total_increase))
25
```

The terminal shows the execution of the code, with the output of the print statements visible at the bottom. The status bar at the bottom of the terminal indicates the file "exec_3.py" is 25 lines long and 5678 bytes written.

Output

```
kamuy@carlosy: ~/wrk/py/univ/assment
kamuy@carlosy: ~/wrk/py/univ/assment x kamuy@carlosy: ~/wrk/py/univ/assment x
kamuy@carlosy:~/wrk/py/univ/assment$ python3 exec_3.py
Enter staff name:
Paul
Enter salary:
1300.00
Enter staff name:
Susan
Enter salary:
1250.00
Enter staff name:
James
Enter salary:
1300.00
Enter staff name:
Rossie
Enter salary:
1350.00
Enter staff name:
Alan
Enter salary:
1400.00

Staff details entered:
Paul          1300.00
Susan         1250.00
James         1300.00
Rossie        1350.00
Alan          1400.00

Total salary amount is: 6600.00
Total salary amount with 10% increase: 7260.00
kamuy@carlosy:~/wrk/py/univ/assment$
```

Bibliography

- [1] D. Hemmendinger, "Computer programming language," Britannica, [Online]. Available: <https://www.britannica.com/technology/computer-programming-language>. [Accessed 25 August 2021].
- [2] P. Documentation, "4. More Control Flow Tools," python.org, 25 August 2021. [Online]. Available: <https://docs.python.org/3/tutorial/controlflow.html>. [Accessed 25 August 2021].
- [3] W3Schools, "Python Looping Through a Range," w3schools.com, 2021. [Online]. Available: https://www.w3schools.com/python/gloss_python_for_range.asp. [Accessed 25 August 2021].
- [4] F. Romano, Learn Python Programming, 2nd Edition, Birmingham, UK: Packt Publishing, 2018.