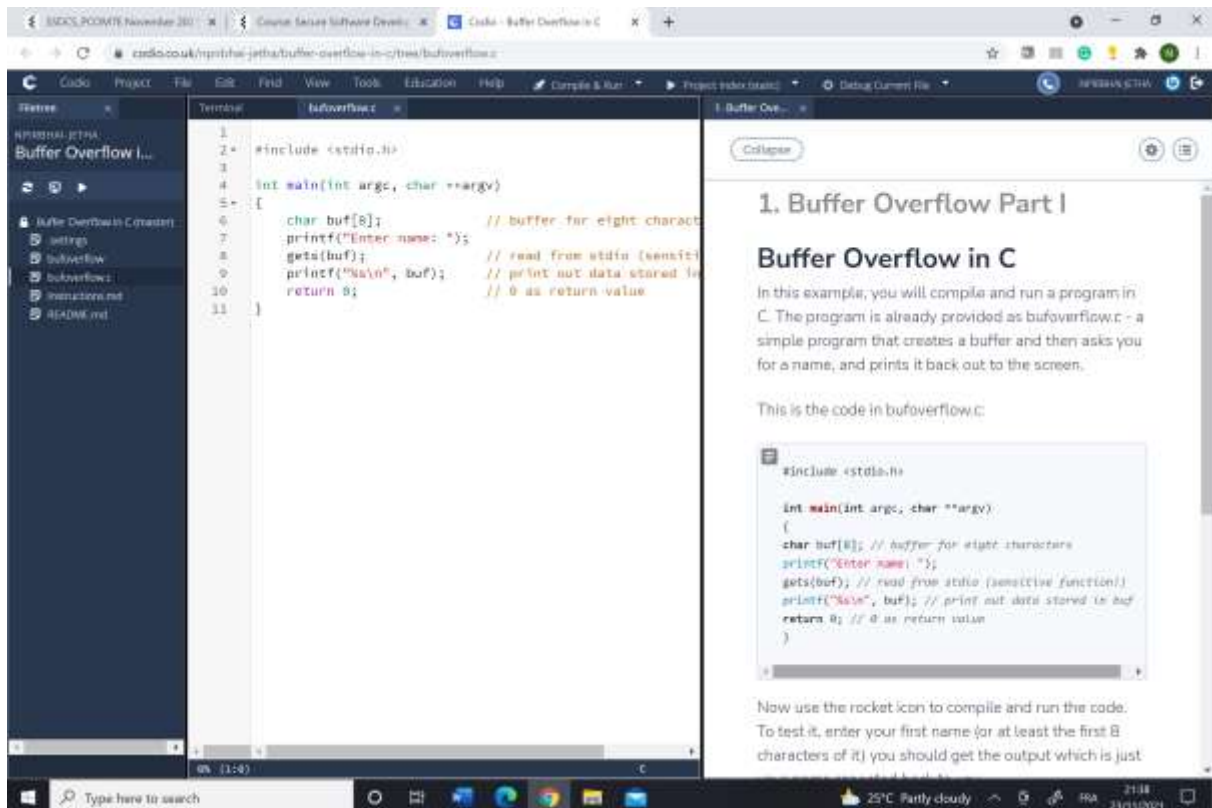


# Codio Activity - Exploring Python tools and features

## Part I

In this example, you will compile and run a program in C using the [Codio workspace](#) provided (Buffer Overflow in C). The program is already provided as `bufoverflow.c` - a simple program that creates a buffer and then asks you for a name, and prints it back out to the screen.



The screenshot displays the Codio IDE interface. On the left, the file explorer shows a project named 'Buffer Overflow I...' with files like 'bufoverflow.c' and 'README.md'. The central code editor shows the following C code:

```
1 #include <stdio.h>
2
3
4 int main(int argc, char **argv)
5 {
6     char buf[8]; // buffer for eight characters
7     printf("Enter name: ");
8     gets(buf); // read from stdin (sensitive function)
9     printf("%s\n", buf); // print out data stored in buf
10    return 0; // 0 as return value
11 }
```

On the right, a documentation panel titled '1. Buffer Overflow Part I' contains the following text:

**Buffer Overflow in C**

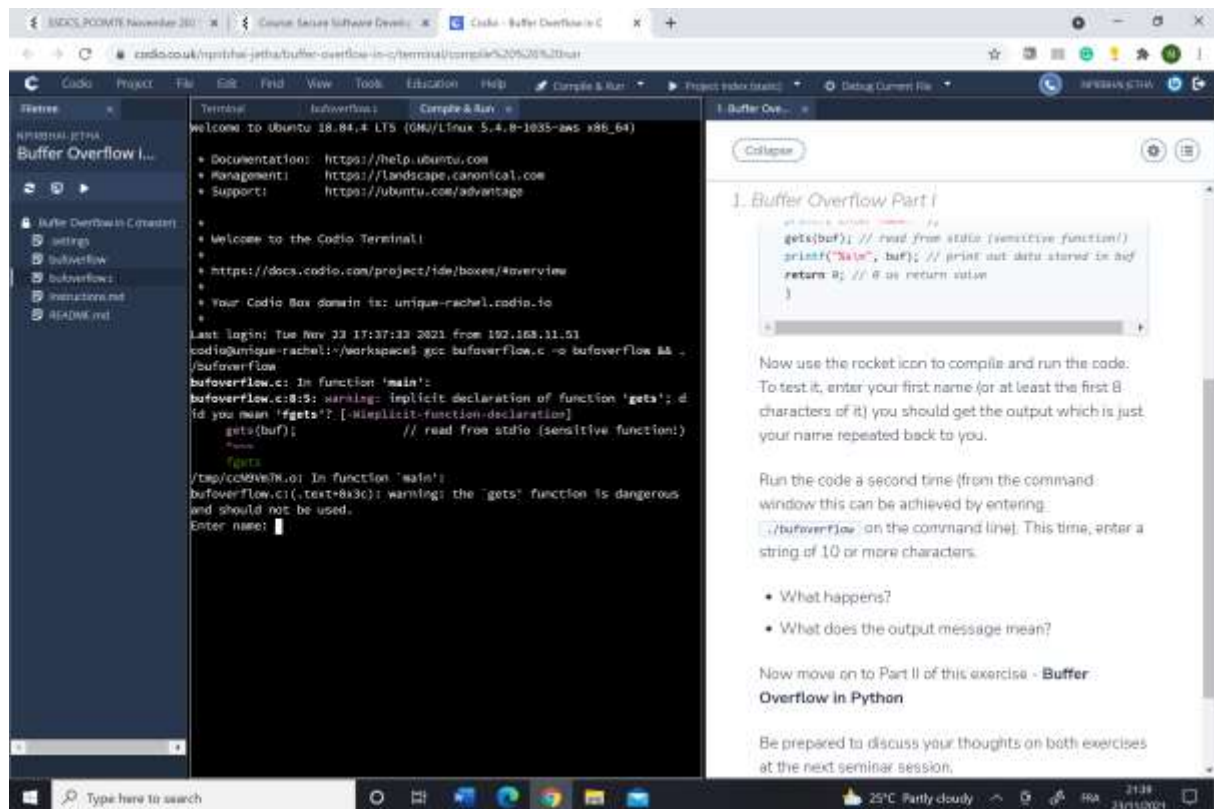
In this example, you will compile and run a program in C. The program is already provided as `bufoverflow.c` - a simple program that creates a buffer and then asks you for a name, and prints it back out to the screen.

This is the code in `bufoverflow.c`:

```
#include <stdio.h>

int main(int argc, char **argv)
{
    char buf[8]; // buffer for eight characters
    printf("Enter name: ");
    gets(buf); // read from stdin (sensitive function)
    printf("%s\n", buf); // print out data stored in buf
    return 0; // 0 as return value
}
```

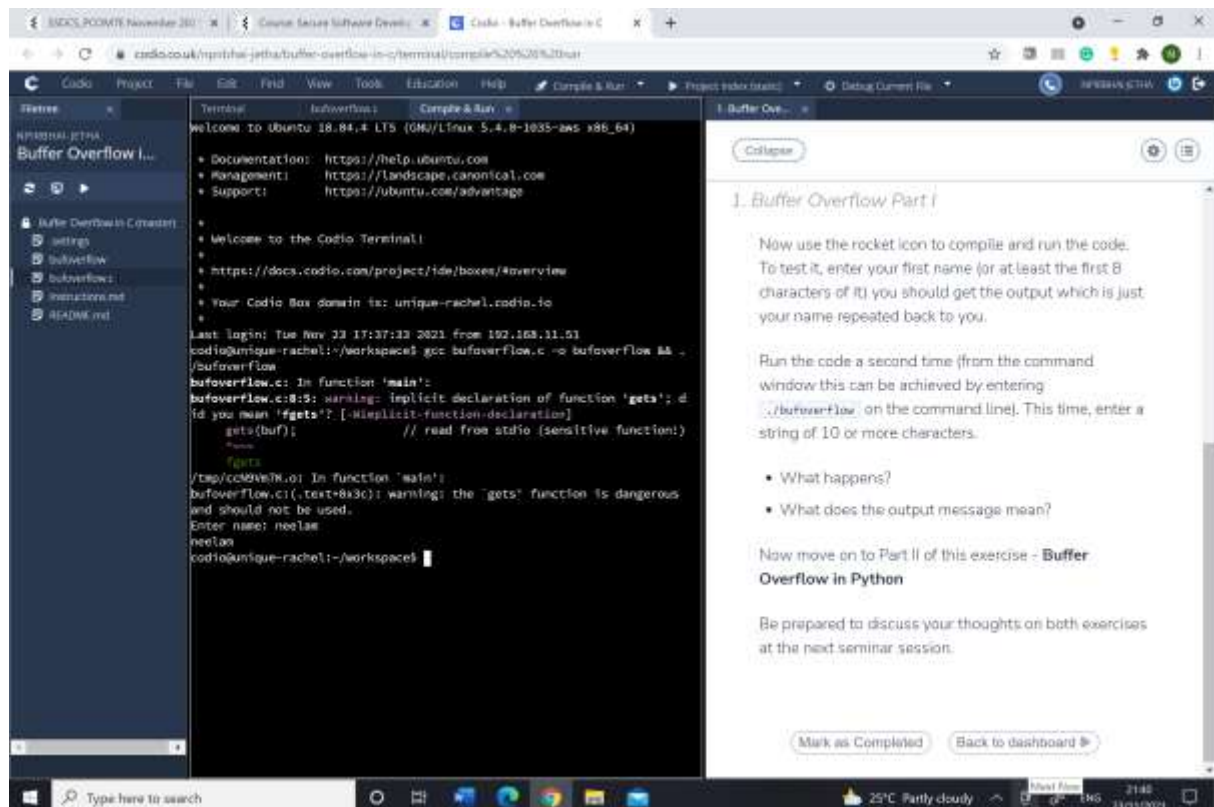
Now use the rocket icon to compile and run the code. To test it, enter your first name (or at least the first 8 characters of it) you should get the output which is just



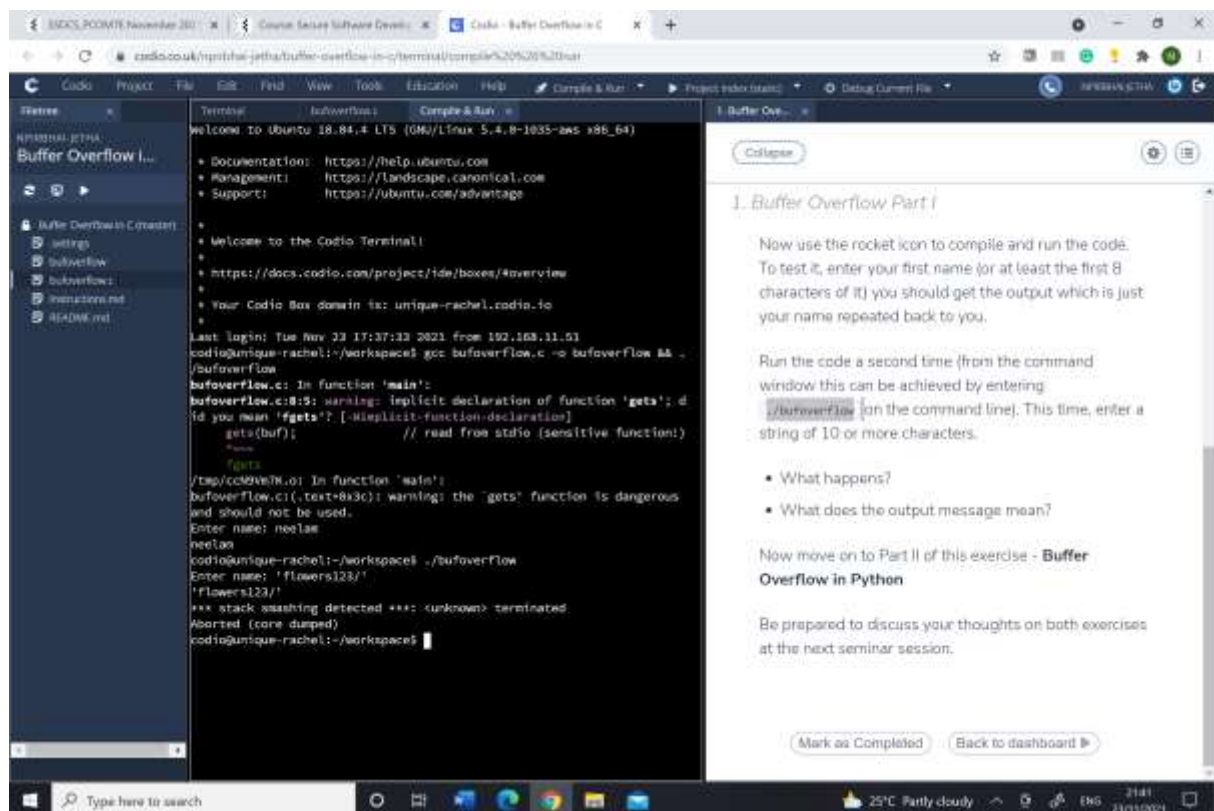
This is the code in `bufoverflow.c` (also available in the Codio workspace):

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[8]; // buffer for eight characters
    printf("enter name:");
    gets(buf); // read from stdio (sensitive function!)
    printf("%s\n", buf); // print out data stored in buf
    return 0; // 0 as return value
}
```

Now compile and run the code. To test it, enter your first name (or at least the first 8 characters of it) you should get the output which is just your name repeated back to you.



Run the code a second time (from the command window this can be achieved by entering `./bufoverflow` on the command line). This time, enter a string of 10 or more characters.



- What happens?

Message received : **stack smashing detected error\*\*\*: <unknown>**,  
terminated, aborted

- What does the output message mean?

“Usually, the compiler generates the **stack smashing detected error** in response to its defense mechanism against buffer overflows.

A **buffer overflow** occurs when the user input exceeds the buffer capacity. The following C code can cause the buffer to overflow if the user enters more than ten characters. In such a case, the compiler will throw the stack smashing detected error.”

[Source: <https://www.educative.io/edpresso/what-is-the-stack-smashing-detected-error>]

The screenshot shows a web browser window with the Codio IDE. The left sidebar shows a file explorer with a project named 'Buffer Overflow'. The main window is split into two panes. The left pane shows a terminal window with the following output:

```

Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/project/ide/boxes/#overview
 *
 * Your Codio Box domain is: unique-rachel.codio.io
 *
Last login: Tue Nov 23 17:37:33 2021 from 192.168.11.51
codio@unique-rachel:~/workspace$ gcc bufoverflow.c -o bufoverflow
bufoverflow.c: In function 'main':
bufoverflow.c:8:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
    gets(buf);           // read from stdin (sensitive function!)
    ^
/tmp/cc09v80k.o: In function 'main':
bufoverflow.c:(.text+0x3c): warning: the 'gets' function is dangerous and should not be used.
Enter name: neelam
neelam
codio@unique-rachel:~/workspace$ ./bufoverflow
Enter name: 'flowers123/'
'flowers123/'
*** stack smashing detected ***: unknown: terminated
Aborted (core dumped)
codio@unique-rachel:~/workspace$

```

The right pane shows a document titled '1. Buffer Overflow Part I' with the following text:

Now use the rocket icon to compile and run the code. To test it, enter your first name (or at least the first 8 characters of it) you should get the output which is just your name repeated back to you.

Run the code a second time (from the command window this can be achieved by entering `./bufoverflow` on the command line). This time, enter a string of 10 or more characters.

- What happens?
- What does the output message mean?

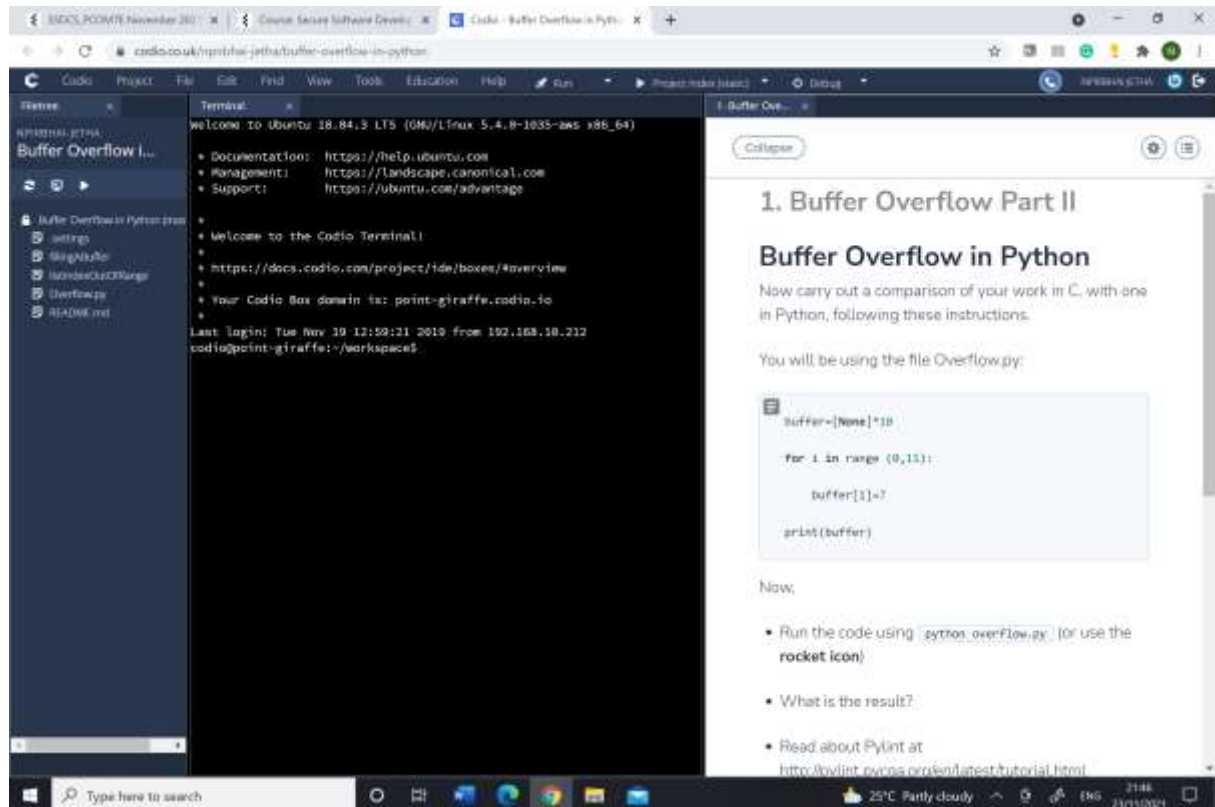
Now move on to Part II of this exercise - **Buffer Overflow in Python**

Be prepared to discuss your thoughts on both exercises at the next seminar session.

At the bottom of the right pane, there are two buttons: 'Mark as Completed' and 'Back to dashboard'.

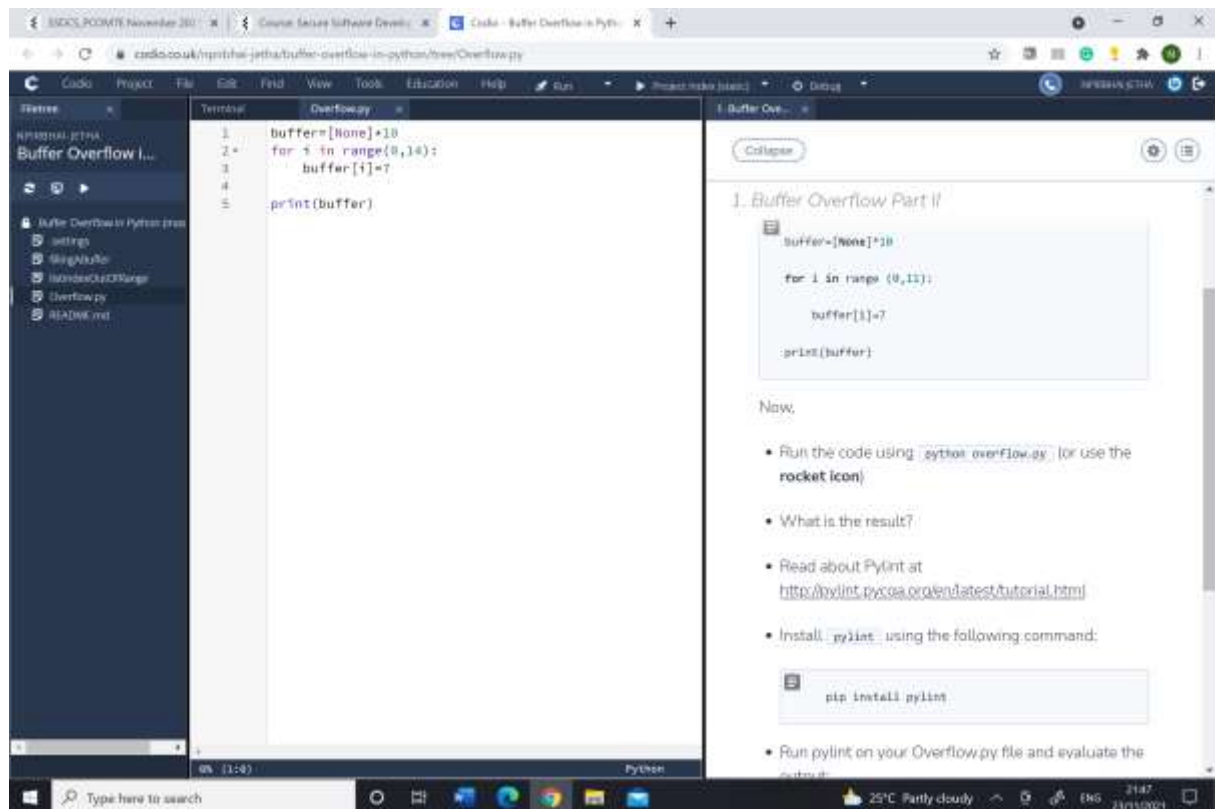
## Part II

Now carry out a comparison of this code with one in Python (Buffer Overflow in Python), following these instructions:



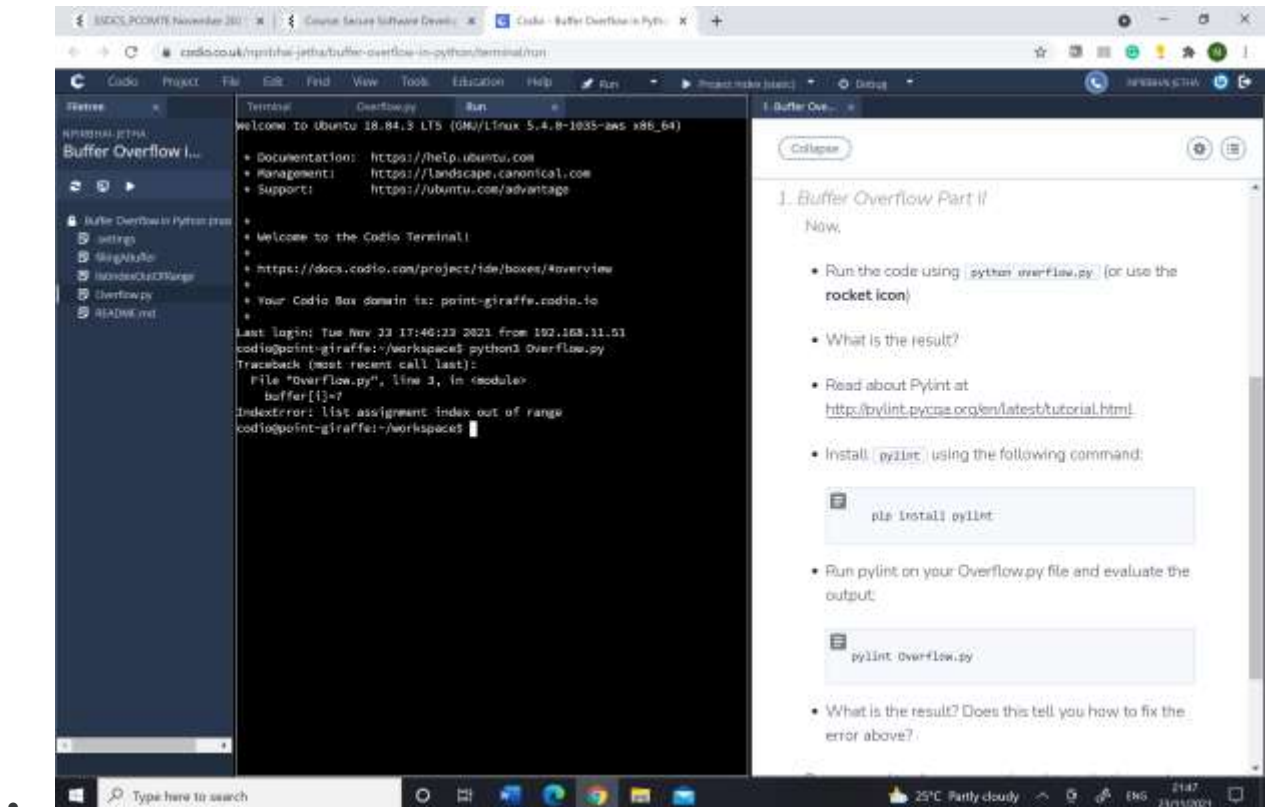
In the Codio workspace, you will be using the file called `Overflow.py`:

```
buffer=[None]*10
for i in range (0,11):
    buffer[i]=7
print(buffer)
```



- Run your code using: Python overflow.py (or use the codio rocket icon)
- What is the result?
- Message received : `IndexError List assignment index out of range`



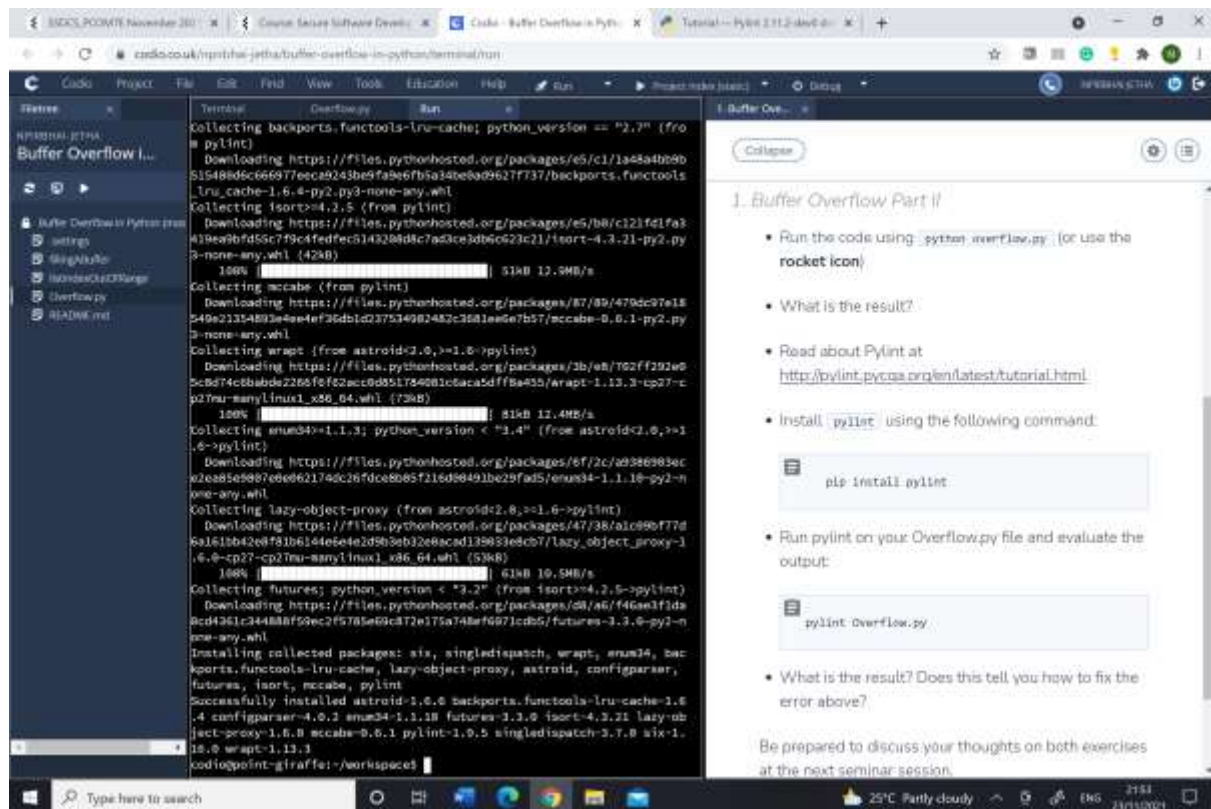


The “`indexerror: list assignment index out of range`” is **raised when you try to assign an item to an index position that does not exist**. To solve this error, you can use `append()` to add an item to a list. You can also initialize a list before you start inserting values to avoid this error.[ <https://careerkarma.com>]

- Read about Pylint at <http://pylint.pycqa.org/en/latest/tutorial.html>

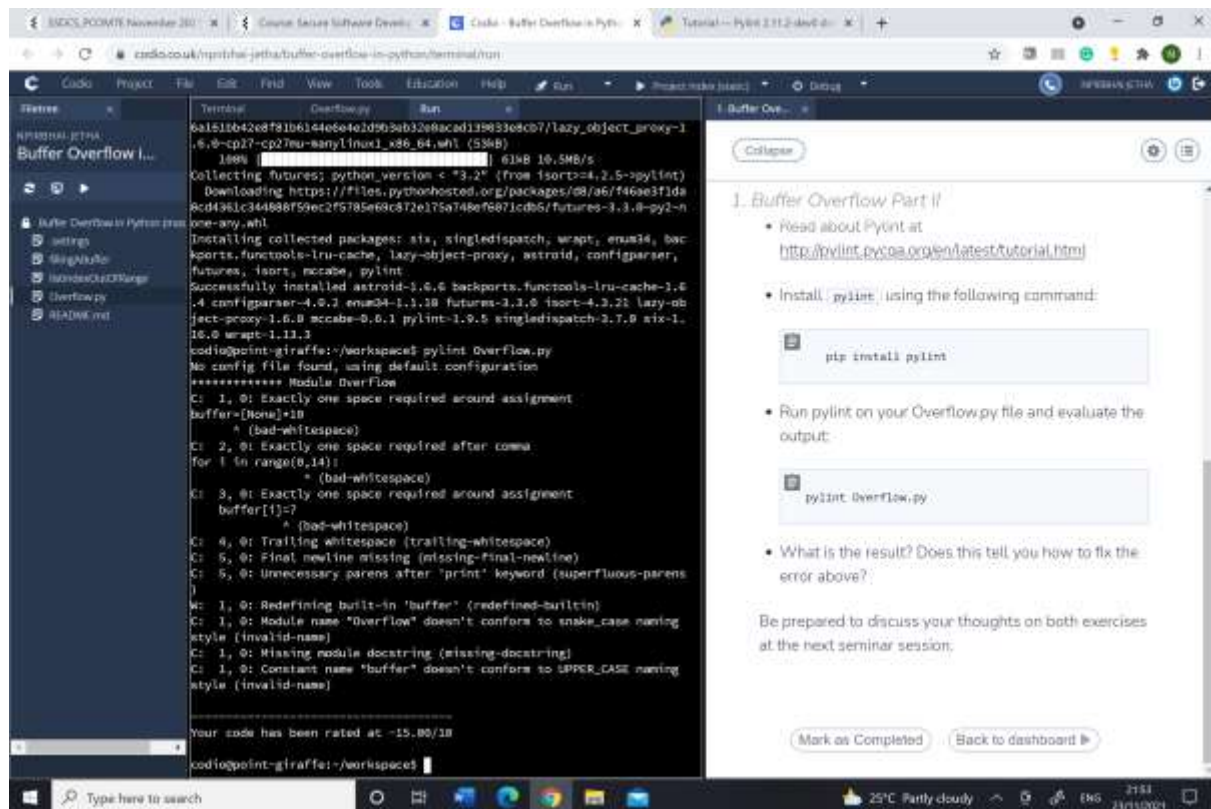
- Install pylint using the following commands:

`pip install pylint` (in the command shell/ interpreter)



- Run `pylint` on one of your files and evaluate the output:  
`pylint your_file`





- (Make sure you are in the directory where your file is located before running Pylint)
- What is the result? Does this tell you how to fix the error above?

It reveals a few of the ‘errors’ and unnecessary actions/codes e.g one space required after comma...

Be prepared to discuss your thoughts on both exercises at next week’s seminar. Remember to record this into your e-portfolio.

### Some additional comments on Pylint

“Pylint is a [source-code](#), [bug](#) and quality checker for the [Python programming language](#). It is named following a common convention in Python of a “py” prefix, and a nod to the C programming [lint](#) program. It follows the style recommended by PEP 8, the Python style guide.<sup>[4]</sup> It is similar to [Pychecker](#) and [Pyflakes](#), but includes the following features:

- Checking the length of each line
- Checking that variable names are well-formed according to the project's coding standard
- Checking that declared interfaces are truly implemented.<sup>[5]</sup>

It is also equipped with the Pyreverse module that allows [UML](#) diagrams to be generated from Python code.

It can be used as a stand-alone program, but also integrates with [IDEs](#) such as [Eclipse](#) with [PyDev](#)<sup>[6]</sup> and [Visual Studio Code](#),<sup>[7]</sup> and editors such as [Atom](#),<sup>[8]</sup> [GNU Emacs](#) and [Vim](#).

It has received favourable reviews.<sup>[9][10][11]</sup>

[source : <https://en.wikipedia.org/wiki/Pylint>]

## Codio Activity: The Producer-Consumer Mechanism

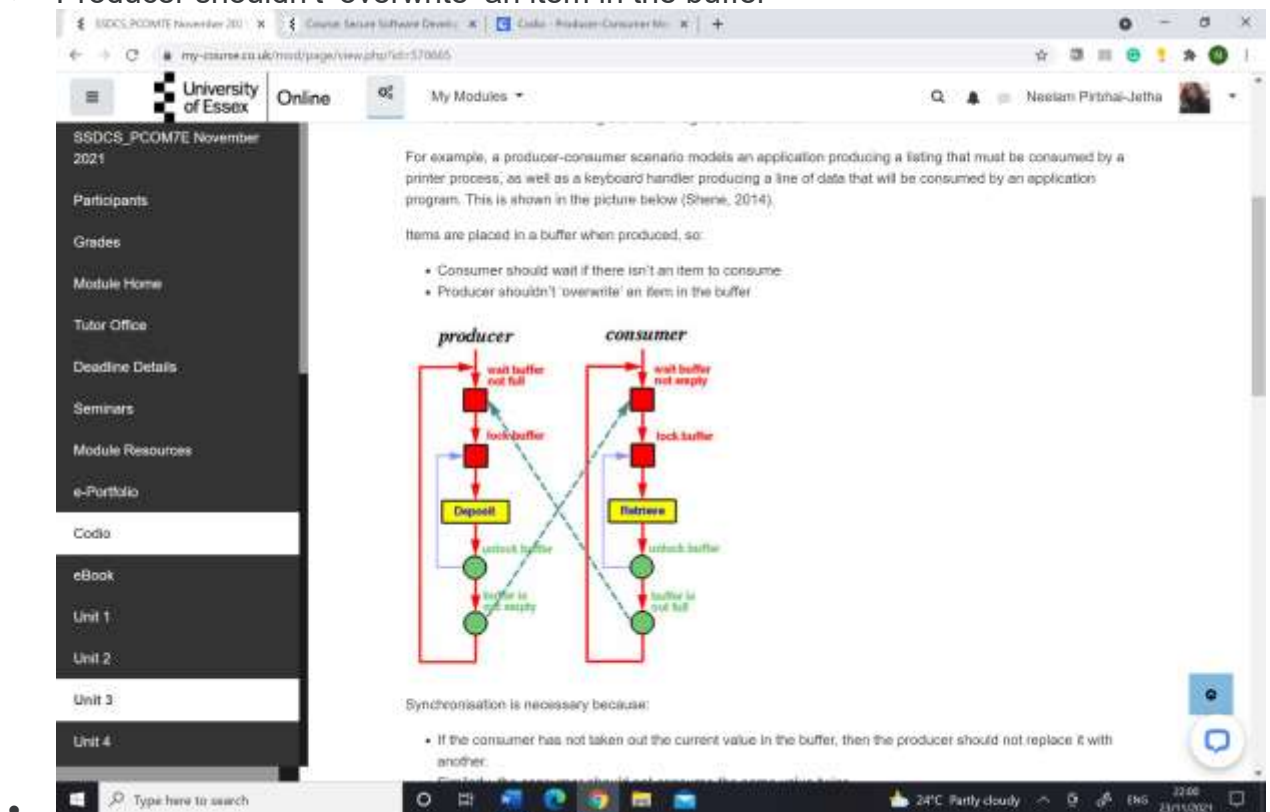
Producer/Consumer Problem (also known as the 'bounded buffer' problem):

- A 'producer' is producing items at a particular (unknown and sometimes unpredictable) rate.
- A 'consumer' is consuming the items – again, at some rate.

For example, a producer-consumer scenario models an application producing a listing that must be consumed by a printer process, as well as a keyboard handler producing a line of data that will be consumed by an application program. This is shown in the picture below (Shene, 2014).

Items are placed in a buffer when produced, so:

- Consumer should wait if there isn't an item to consume
- Producer shouldn't 'overwrite' an item in the buffer

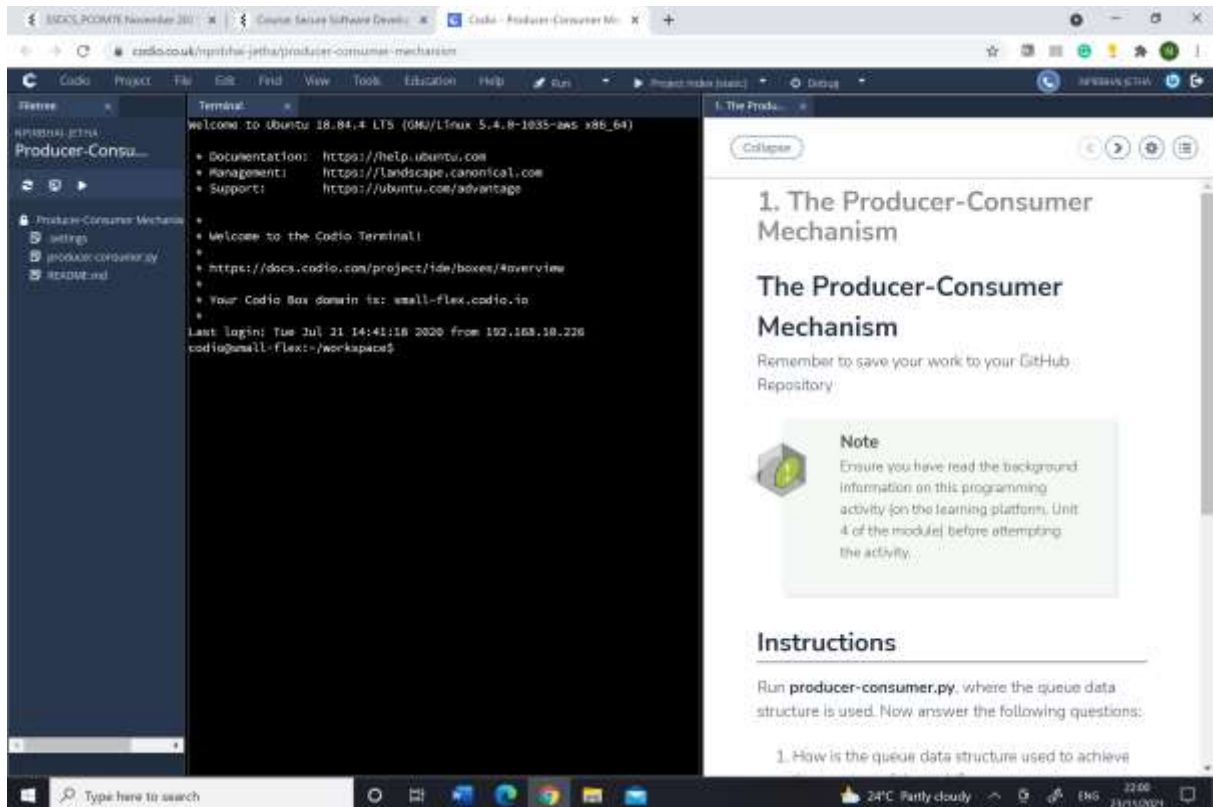


Synchronisation is necessary because:

- If the consumer has not taken out the current value in the buffer, then the producer should not replace it with another.
- Similarly, the consumer should not consume the same value twice.

## Task

Run `producer-consumer.py` in the provided Codio workspace (**Producer-Consumer Mechanism**), where the queue data structure is used.



A copy of the code is available here for you.

# code source: <https://techmonger.github.io/55/producer-consumer-python/>

```
from threading import Thread
from queue import Queue

q = Queue()
final_results = []

def producer():
    for i in range(100):
        q.put(i)

def consumer():
    while True:
        number = q.get()
        result = (number, number**2)
        final_results.append(result)
        q.task_done()

for i in range(5):
    t = Thread(target=consumer)
    t.daemon = True
```

```

t.start()

producer()

q.join()

print (final_results)

```

The top screenshot shows the Codio IDE interface with the following code in the editor:

```

1 # code source: https://technonger.github.io/55/producer-consumer/
2
3 from threading import Thread
4 from queue import Queue
5
6 q = Queue()
7 final_results = []
8
9 def producer():
10     for i in range(100):
11         q.put(i)
12
13 def consumer():
14     while True:
15         number = q.get()
16         result = (number, number**2)
17         final_results.append(result)
18         q.task_done()
19
20 for i in range(5):
21     t = Thread(target=consumer)
22     t.daemon = True
23     t.start()
24
25 producer()
26
27 q.join()
28
29 print (final_results)

```

The bottom screenshot shows the same IDE after running the code. The terminal output is as follows:

```

Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.4.0-1035-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/advantage

 * Welcome to the Codio Terminal!
 * https://docs.codio.com/project/ide/boxes/4/overview
 * Your Codio Box domain is: small-flex.codio.io

Last login: Tue Nov 22 15:00:28 2021 from 192.168.11.51
codio@small-flex:~/workspace$ python3 producer-consumer.py
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100), (11, 121), (12, 144), (13, 169), (14, 196), (15, 225), (16, 256), (17, 289), (18, 324), (19, 361), (20, 400), (21, 441), (22, 484), (23, 529), (24, 576), (25, 625), (26, 676), (27, 729), (28, 784), (29, 841), (30, 900), (31, 961), (32, 1024), (33, 1089), (34, 1156), (35, 1225), (36, 1296), (37, 1369), (38, 1444), (39, 1521), (40, 1600), (41, 1681), (42, 1764), (43, 1849), (44, 1936), (45, 2025), (46, 2116), (47, 2209), (48, 2304), (49, 2401), (50, 2500), (51, 2601), (52, 2704), (53, 2809), (54, 2916), (55, 3025), (56, 3136), (57, 3249), (58, 3364), (59, 3481), (60, 3600), (61, 3721), (62, 3844), (63, 3969), (64, 4096), (65, 4225), (66, 4356), (67, 4489), (68, 4624), (69, 4761), (70, 4900), (71, 5041), (72, 5184), (73, 5329), (74, 5476), (75, 5625), (76, 5776), (77, 5929), (78, 6084), (79, 6241), (80, 6400), (81, 6561), (82, 6724), (83, 6889), (84, 7056), (85, 7225), (86, 7396), (87, 7569), (88, 7744), (89, 7921), (90, 8100), (91, 8281), (92, 8464), (93, 8649), (94, 8836), (95, 9025), (96, 9216), (97, 9409), (98, 9604), (99, 9801)]
codio@small-flex:~/workspace$

```

Answer the following questions:

1. How is the queue data structure used to achieve the purpose of the code?

Queue, as the name suggests is used **whenever we need to manage any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn.**

2. What is the purpose of `q.put()`?

`Queue.put(item, block=True, timeout=None)`<sup>1</sup>

Put *item* into the queue. If optional args *block* is true and *timeout* is `None` (the default), block if necessary until a free slot is available. If *timeout* is a positive number, it blocks at most *timeout* seconds and raises the `Full` exception if no free slot was available within that time. Otherwise (*block* is false), put an item on the queue if a free slot is immediately available, else raise the `Full` exception (*timeout* is ignored in that case).

[<https://docs.python.org/3/library/queue.html>]

3. What is achieved by `q.get()`?

`Queue.get(block=True, timeout=None)`<sup>1</sup>

Remove and return an item from the queue. If optional args *block* is true and *timeout* is `None` (the default), block if necessary until an item is available. If *timeout* is a positive number, it blocks at most *timeout* seconds and raises the `Empty` exception if no item was available within that time. Otherwise (*block* is false), return an item if one is immediately available, else raise the `Empty` exception (*timeout* is ignored in that case).

Prior to 3.0 on POSIX systems, and for all versions on Windows, if *block* is true and *timeout* is `None`, this operation goes into an uninterruptible wait on an underlying lock. This means that no exceptions can occur, and in particular a `SIGINT` will not trigger a `KeyboardInterrupt`.

[<https://docs.python.org/3/library/queue.html>]

4. What functionality is provided by `q.join()`?

Blocks until all items in the queue have been gotten and processed.

The count of unfinished tasks goes up whenever an item is added to the queue. The count goes down whenever a consumer thread calls `task_done()` to indicate that the item was retrieved and all work on it is complete. When the count of unfinished tasks drops to zero, `join()` unblocks.

[<https://docs.python.org/3/library/queue.html>]

5. Extend this producer-consumer code to make the producer-consumer scenario available in a secure way. What technique(s) would be appropriate to apply?

**Remember** to record your thoughts and answers in your e-portfolio.

### **Learning Outcomes**

- Identify and critically analyse operating system risks and issues, and identify appropriate methodologies, tools and techniques to solve them.
- Critically analyse and evaluate solutions produced.