



# Code Afrique 2K19

Kumasi and Accra, Ghana  
January 2019



# Ice Breakers

Attention!!!!

Make sure you have registered outside



# Introductions

Video Introductions



# Introductions

Professors and Mentors

# Sponsors & Partners



Google



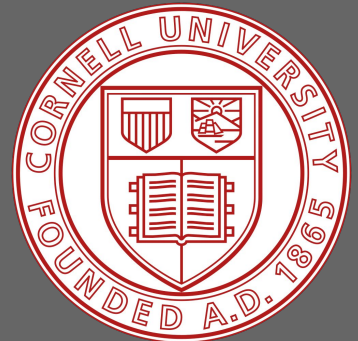
facebook®




slack



**SIGOPS**





# What is and Why Pursue Computer Science?



# What is Computer Science?

*CS = Problem solving using computers*

- Machine Learning and Artificial Intelligence (AI)
- Data Science, Big Data
- Theory and Algorithms
- Programming Languages
- Scientific Computing
- Human Computer Interfacing
- Systems, Networking, and Databases
- Security
- Graphics
- Architecture
- Robotics
- Computational Biology
- Quantum Computing
- ...

*It's not just programming!!*



# What is CS Used For?

- Laptops, Mobile Phones, etc.
- Word processing, spreadsheets, etc.
- Computer games, computer animation
- Self-driving cars and trucks, robots, ...
- The “Web”
- Apps and “The Cloud”
  - Google, Facebook, WhatsApp, Instagram, ...
  - Uber, Amazon, eBay, ...
- Blockchains and Cryptocurrencies
- Smart home, smart farm, smart city, ...
- Air Traffic Control, Power Grid, etc.
- Modern healthcare
- much much more

*What are your favorite computer apps?*





# Careers in Computer Science

- Become a programmer
  - specialize in robots, computer games, or ...
- Become a web designer
  - build nice websites
- Become a data analyst
  - collect data and figure out what it means
- Become a systems/network/database administrator
  - maintains systems, email, computer networks, etc.
- Become an entrepreneur
  - start a company (consulting, new tech, etc.)
- Become a professor
  - come up with new uses for computers, and teach students about CS
- ...



## Summary

- There are many high paying CS jobs out there!!
- Almost every industry needs CS experts
- Working with all kinds of people on all kinds of problems
- International opportunities



BREAK



Find your teammates



# Command line Basics

# Command Line Basics

Windows:

- `cd ..`
- `dir`

Linux:

- `cd ..`
- `ls`



# Intro To Python

# The Python Shell

Type python and hit enter

```
bash: gcc: command not found
[Samuels-MacBook-Pro:~ samuel-personal$ python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:04:33)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```



# Spyder

The image shows the Spyder Python IDE interface. The main window is titled "Editor - /Users/rvr/.spyder-py3/temp.py" and contains a code editor with the following Python code:

```
1# -*- coding: utf-8 -*-
2"""
3Spyder Editor
4
5This is a temporary script file.
6"""
7
8print("hello")
```

The code editor has a light blue background and a line number column on the left. The code is color-coded: comments are grey, docstrings are green, and the print statement is purple and green. The line number 8 is highlighted in light blue.

Below the code editor is the IPython console, which displays the output of the code:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rvr/.spyder-py3/temp.py', wdir='/Users/
rvr/.spyder-py3')
hello

In [2]:
```

On the right side of the interface, there is a "Help" window titled "Usage". It contains the following text:

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

At the bottom of the interface, there are three tabs: "Variable explorer", "File explorer", and "Help". The "Help" tab is currently selected.

# Spyder

The image shows the Spyder Python IDE interface. The main window is titled "Editor - /Users/rvr/.spyder-py3/temp.py". The code editor contains the following text:

```
1# -*- coding: utf-8 -*-
2"""
3Spyder Editor
4
5This is a temporary script file.
6"""
7
8print("hello")
```

The line `8print("hello")` is circled in red. The right-hand side of the interface is split into two panels. The top panel is titled "Help" and shows a "Usage" section with the following text:

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

The bottom panel is titled "IPython console" and shows the following output:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rvr/.spyder-py3/temp.py', wdir='/Users/
rvr/.spyder-py3')
hello

In [2]:
```

# Spyder

temp.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8 print("hello")
```

Usage

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Variable explorer File explorer Help

Console 1/A

Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rvr/.spyder-py3/temp.py', wdir='/Users/rvr/.spyder-py3')  
hello

In [2]:

# Spyder

The image shows the Spyder Python IDE interface. The main window is titled "temp.py" and contains the following code:

```
1# -*- coding: utf-8 -*-
2"""
3Spyder Editor
4
5This is a temporary script file.
6"""
7
8print("hello")
```

The code on line 8 is highlighted in a light purple color. To the right of the editor, there is a "Help" panel with the following text:

**Usage**

Here you can get help of any object by pressing **Cmd+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Below the help panel, there are tabs for "Variable explorer", "File explorer", and "Help".

At the bottom, there is an "IPython console" window. The console output is as follows:

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/rvr/.spyder-py3/temp.py', wdir='/Users/
rvr/.spyder-py3')
hello
In [2]:
```

The word "hello" in the console output is circled in red.

# Print Statements



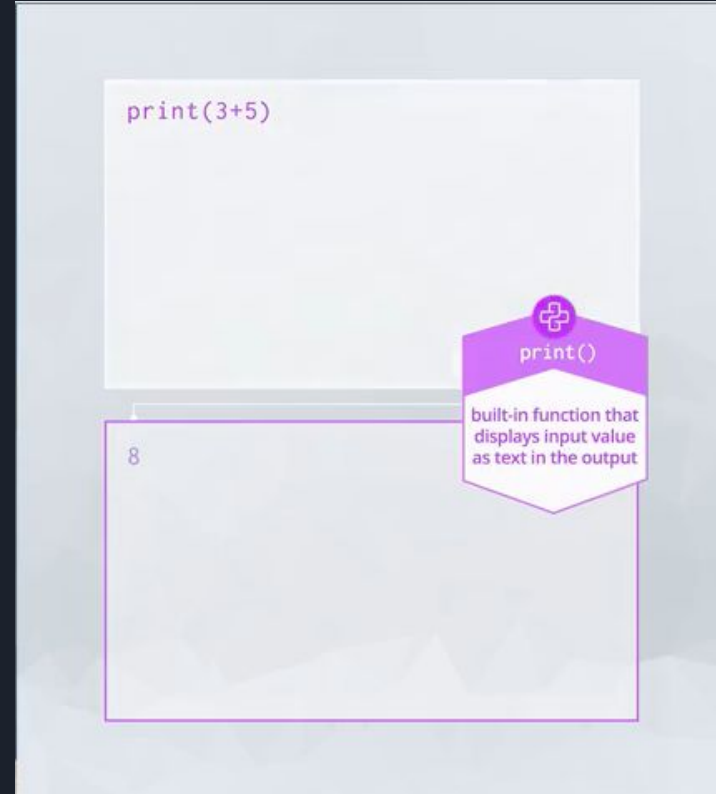
# Print Statements

To output something in Python, we use print statements.

In Python 3, we use the syntax:

- `print("Statement to be printed")`

E.g.: `print("Hello World!")`



# With Spyder

The image shows the Spyder Python IDE interface. On the left is the source code editor for a file named `temp.py`. The code contains a docstring, a title, a description, and a print statement.

```
1# -*- coding: utf-8 -*-  
2''''  
3Spyder Editor  
4  
5This is a temporary script file.  
6''''  
7  
8print("hello")
```

On the right is the IPython console, which shows the execution of the code. The first execution (In [1]) runs the script, resulting in the output "hello". The second execution (In [2]) runs the print statement, resulting in the output "Hello World". This second execution and its output are circled in red.

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)  
Type "copyright", "credits" or "license" for more information.  
  
IPython 7.2.0 -- An enhanced Interactive Python.  
  
In [1]: runfile('/Users/rvr/.spyder-py3/temp.py', wdir='/Users/  
rvr/.spyder-py3')  
hello  
  
In [2]: print("Hello World")  
Hello World  
  
In [3]:
```



# Question on Print Statements

- Print your name
- Print your favourite food





# Operators

# Arithmetic Operators

ARITHMETIC OPERATORS	
SYMBOL	OPERATION
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
%	modulo
//	integer division

# OPERATORS

```
#Calculating in Modulo
```

```
print (7 + 5)
```

```
>> 12
```

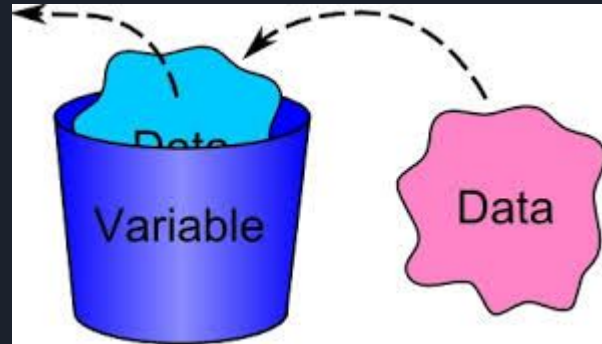
# OPERATORS

```
#Calculating in Modulo
```

```
print (7 % 5)
```

```
>> 2
```

# Variables





# Variables

A *variable* is a name that stores a value.

An *assignment statement* associates a variable name on the left side of the equal sign with the value on the right side.


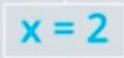
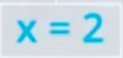

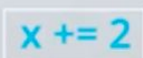


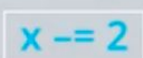
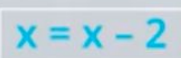

Examples of variable assignment:

```
cat = 5;
```

```
dog = 82;
```

# Variables and Assignments

## • ASSIGNMENT OPERATORS •

SYMBOL	EXAMPLE	EQUIVALENT
		
		
		
		

and many more!

# ASSIGNMENT OPERATORS

```
#Assigning variables
```

```
population = 23456
```

```
population += 400 - 56
```

```
>> 23800
```





# Data Types in Python

- **int** e.g. 1, -3, 5, -10
- **float** (decimal) e.g. 2.3,3.0,4.5,5.7 etc
- **string** e.g.: 'Cynoc', 'Sena' , 'Samuel', 'Jesse'
- **boolean**: True or False



# Boolean examples

$3 == 3$  evaluates to True

$3 == 4$  evaluates to False

$3 != 3$  evaluates to False

$3 != 4$  evaluates to True

True or False evaluates to True

True and False evaluates to False

not True evaluates to False



# Working with Strings



# What really are strings?

- In computer science, a string is any finite sequence of characters.

i.e. letters, numerals, symbols, and punctuations marks

Strings are quoted characters:

1. “ab cd” (most languages use this)



# So then How do you represent quotes in quotes?

You can do this in 3 ways:

- You can escape the quotes using backslash (\):
  - `print “\” A word that needs quotations marks\””`
- Use single and double quotes together:
  - `print ‘ “A word that needs quotation marks” ‘`
- Use triple-quoted strings:
  - `print “”” “A word that needs quotation marks” “””`



# Strings can be indexed

`S='abc d'`

Access characters of string with the symbol `[]` and strings are zeroed indexed:

`S[0]` is 'a', `S[1]` is 'b', and so on

Negative indexes are from the end:

`S[-1]` is 'd', `S[-5]` = 'a'



# Strings can be “sliced”

`S='abc d'`

`S[0:3]` is 'abc'

`S[2:5]` is 'c d'

`S[1:]` is 'bc d'

`S[:2]` is 'ab'



# Question

S='Hello all'

What is S[3:6] ?

A='llo'

B='lo '

C='o a'

D='o '

E=Sorry bro, No idea





# Question

S='Hello all'. What is S[4:] ?

A='Hell'

B='o all'

C='Hello'

D=Error!

E= Sorry bro, No idea

What about S[:4] ?



# Other cool stuffs you can do with strings!!!

You can check if a substring (part) of the string equals to another string. This evaluates to a boolean (True or False).

Eg: `s='cadasdsaxasaadsda'`

“`'cad' in s`” evaluates to True while “`'foo' in s`” evaluates to false



# Cool method for strings

A cool function for string is “len” which returns the number of chars in a String.

Eg. `len(s)=17`

`len(s[3:8])=5` etc

Quick Question

`len(s[3:8])==6` ( True or False)



## Other Cool Methods associated with Strings:

Use the dot syntax when calling a method unique to Strings.

Example: `s= 'Hello World'`

`s.index('e')=1`

`s.count('l')=3`

`s.upper() == 'HELLO WORLD'`

Resource for finding other cool methods on

strings=<https://docs.python.org/3/library/stdtypes.html#str>



# Had Enough of Strings? One more fact lol

Strings can be concatenated (meaning that you can add two or more strings together to form one string)

Example:

```
Owass= 'Cynoc, Jesse, '
```

```
SOS= 'Sena'
```

```
Labone= 'Samuel'
```

```
message= 'love to code and we love you all'
```

```
Ghana_hacks=Owass + Labone + 'and' + SOS + message
```



# Lists



# Introduction to Lists

What is a list?

- A mutable data structure used to order the sequence of elements.
- In Python, lists are denoted by `[ ]`, and the values within are separated by commas.
- Lists can be assigned to variables.



## Examples of Lists

E.g.: `x = [3, "Hello", 9, True]`

`result = [4, 0, 3, 5, 3, 6]`

- NB: Lists can contain any type at any position





# String-Like Properties of Lists

- Lists are zero-indexed like strings.
- Elements in a list can be accessed using [ ].

Eg: `x = [6, 2, 9, "People"]`

`x[0] = 6`, `x[3] = "People"`, `x[4]` will give an error, `x[-1] = "People"`

- Lists can also be sliced. Slicing *always* creates a *copy* of the list.

Eg. If `b = x[0:2]`, then `b` is `[6, 2]` and `x` is unchanged.



## Useful List Methods

**list.append(x)** - Adds x to the end of the list.

**list.extend(iterable)** - Extend the list by appending all the items from the iterable. An iterable is any type that can be looped over; a list is an example of such.

**list.index(x)** - Finds the position of x in the list, gives error if x is not present.



## More Useful List Methods

**list.sort()** - Sorts the list. Additional parameters can be given to modify how the list is sorted.

**list.insert(i, x)** - Inserts item  $x$  at position  $i$  in the list.

More of these methods can be found in the Python documentation at

<https://docs.python.org/3/tutorial/datastructures.html>



# Using external libraries (Modules)



# External libraries

- External libraries or modules contain functions that we can use without having to re-implement those functions.
- To use an external library in Python, we first have to import that external library.
- We can then call functions from the external library.



# Using the Math Library

# MATH LIBRARY/MODULE

```
import math
```

```
print math.sqrt(9)
```

```
>> 3
```



# IF STATEMENTS





# IF STATEMENT

```
#create username and password
username = "Sam"
Password = "IamCool"

If name == "Sam":
    print "Sam is logging in"

If name != "Sam":
    print "A hacker is logging in"
```

# IF-ELSE STATEMENT

```
#create username and password
username = "Sam"
Password = "IamCool"

If name == "Sam":
    print "Sam is logging in"
else:
    print "Password Incorrect"
```

# IF-ELIF- ELSE STATEMENT

```
#create username and password
username = "Sam"
Password = "IamCool"

if name == "Sam":
    print "Sam is logging in"
elif name == "John":
    print "John is logging in"
else:
    print "Password Incorrect"
```



# Loops



# Introduction to Loops

- Loops are mainly used to process sequences/iterables such as strings and lists
- Sidenote: The range function

Python's `range()` function generates an iterator (not a list!) of numbers from 0 up to the number exclusively, and is generally iterated over in `for`-loops.



# Loops continued

`range(5)` stores `[0, 1, 2, 3, 4]`

`range(0, 6)` stores `[0, 1, 2, 3, 4, 5]`

`range(4, 9, 2)` stores `[4, 6, 8]`

`list(range(5)) = [0, 1, 2, 3, 4]`

You'll see one practical application of this in a bit.



# For-Loops

```
for x in iterable:
```

```
    Do something
```

General structure of for-loop  
Iterable is the loop sequence  
x is the loop variable  
Do something is the body

So if I want to print every value in my list *a* for example, I would write something like:

```
for x in a:
```

```
    print(x)
```

This will print all the values in a



# Another Way to Do For-Loops

```
for x in range(10):
```

```
    print(x + 5)
```

What would this do?

Practical way of using for-loops like this:

If I have some string, and I want to process each letter of the string, I could write

```
for y in range(len(string)):
```

```
    Do something to string[y]
```

- Note: You choose which way to implement your for-loop depending on the situation you need them for. Sometimes one method is easier than the other.



# While-Loops

- We can only use a for-loop if we know the number of times we want to do something.
- For-loops cannot handle cases where we want to loop an unknown number of times.
- While-loops can handle these cases, however while loops are trickier to use.

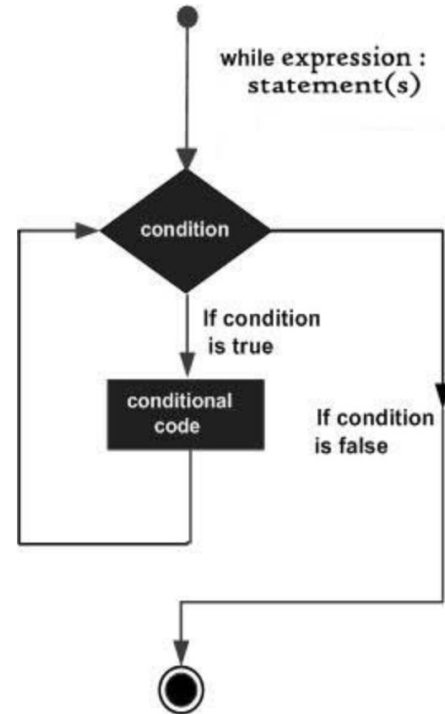
# While Loop Syntax

while <condition>:

Do stuff

The loop stops when the condition is false. However *you* need to manage this explicitly in your code, otherwise you will create an *infinite loop*!

Flow Diagram





# While-Loops

Eg.

`n = 0`

`m = 7`

`while n < m:`

`print("n is less than m")` ← *Body*

`...`


`n = n + 1`

*What happens if this last line isn't there?*

# While-Loops

```
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1

print "Good bye!"
```



# While-Loops

The count is: 0

The count is: 1

The count is: 2

The count is: 3

The count is: 4

The count is: 5

The count is: 6

The count is: 7

The count is: 8

Good bye!



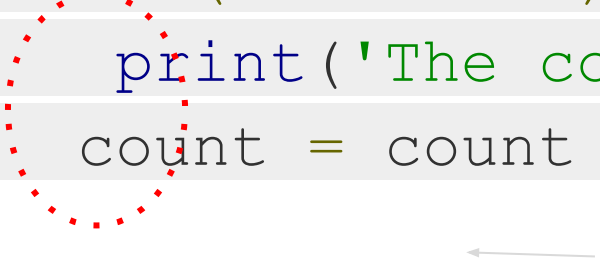
# Note on indentation

```
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1
print "Good bye!"
```

**correct indentation is very important!!**

# Note on indentation

```
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1
print "Good bye!"
```



this won't work

# Note on indentation

```
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1

    ←
print "Good bye!"
```

what does this do?



# Note on indentation

```
count = 0
while (count < 9):
    print('The count is:', count)
    count = count + 1
print "Good bye!"
```

this is the right way

# Infinite Loops

```
var = 1
while var == 1 : # This constructs an
infinite loop
    num = input("Enter a number :")
    print("You entered: ", num)

print "Good bye!"
```

# Infinite Loops

```
Enter a number :20
```

```
You entered: 20
```

```
Enter a number :29
```

```
You entered: 29
```

```
Enter a number :3
```

```
You entered: 3
```

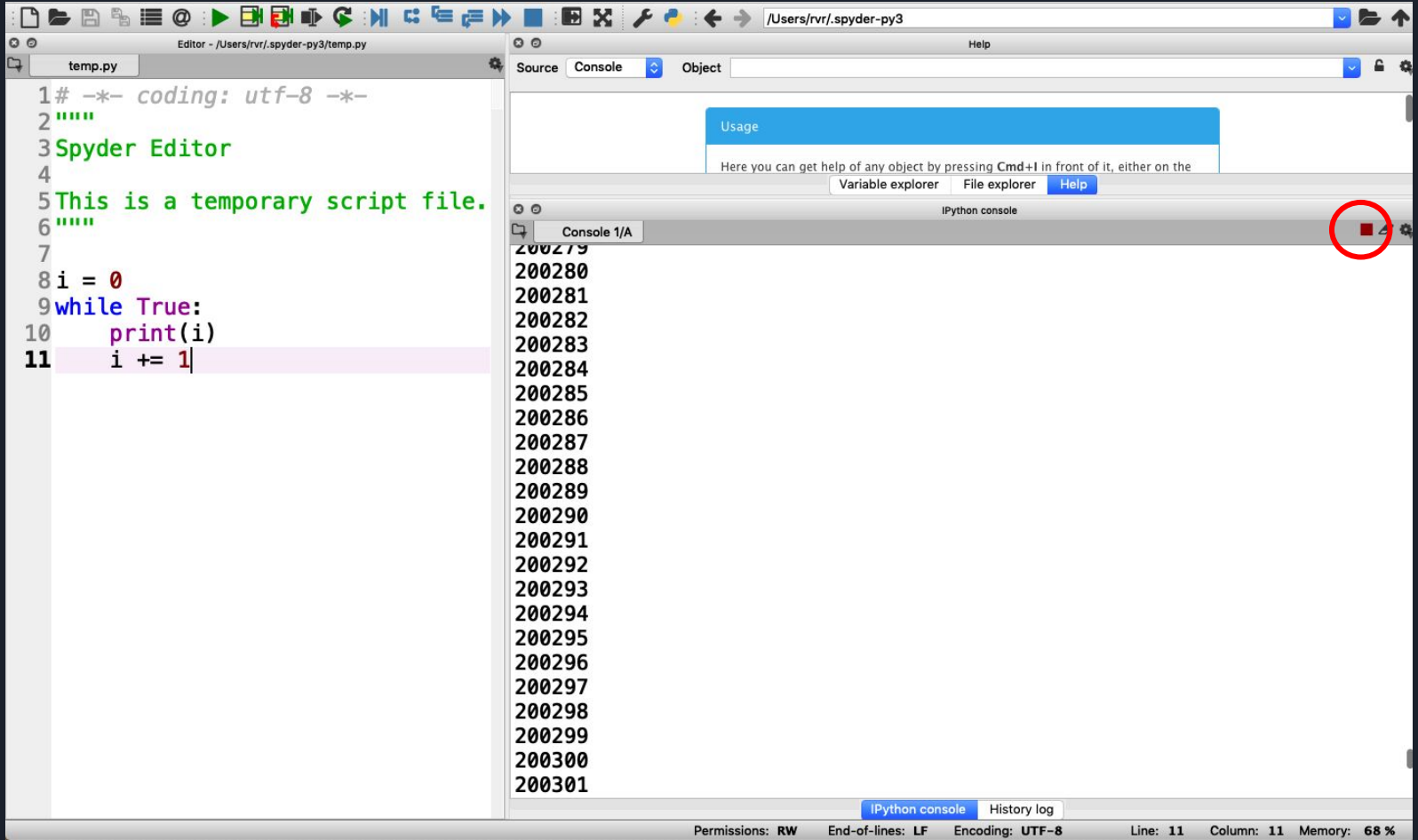
```
Enter a number between :Traceback (most recent call  
last):
```

```
File "test.py", line 5, in <module>
```

```
    num = raw_input("Enter a number :")
```

```
KeyboardInterrupt
```

# Infinite loop in Spyder



The screenshot displays the Spyder IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

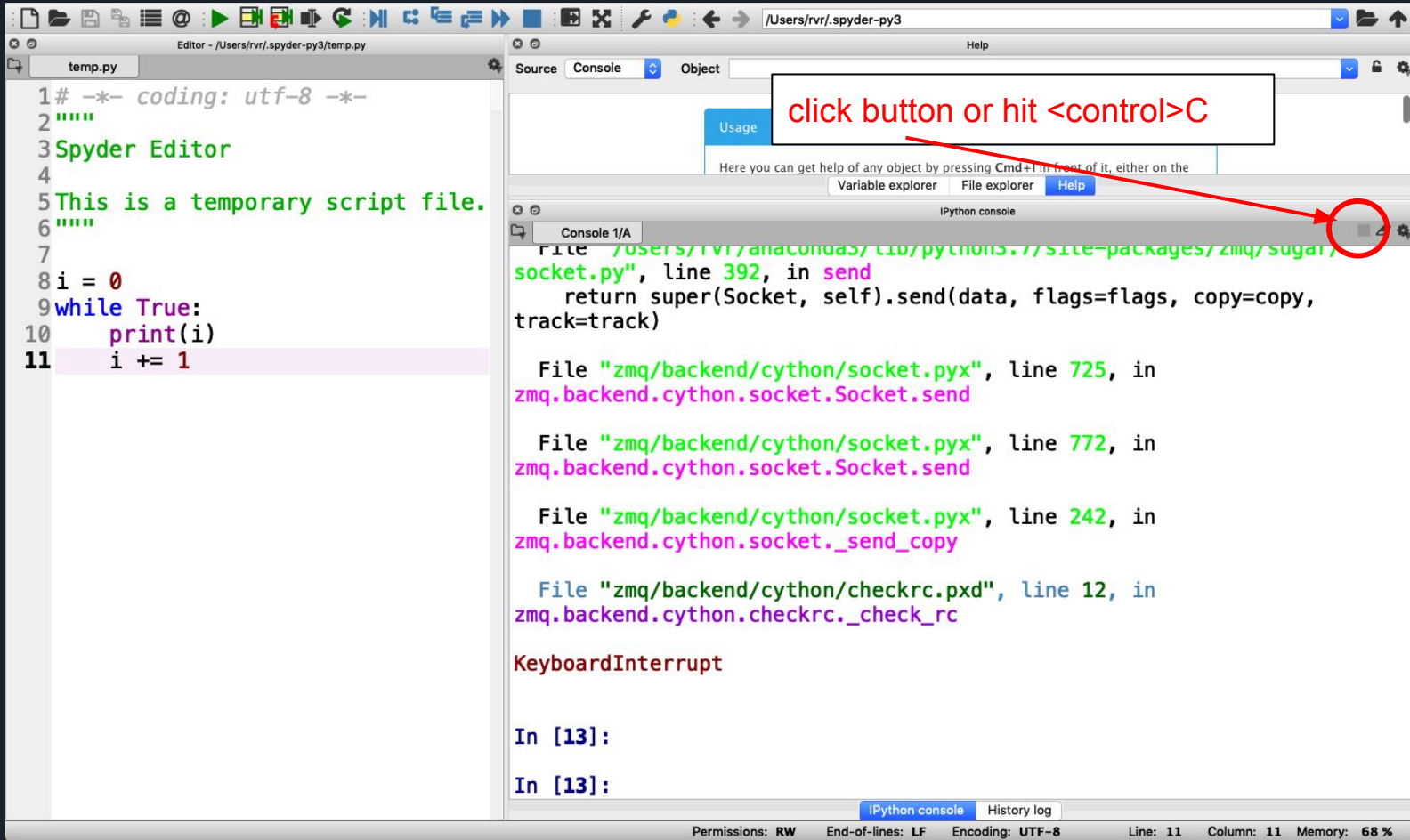
```
1# -*- coding: utf-8 -*-
2"""
3Spyder Editor
4
5This is a temporary script file.
6"""
7
8i = 0
9while True:
10    print(i)
11    i += 1
```

The code is designed to create an infinite loop. The `while True:` loop contains a `print(i)` statement followed by `i += 1`. The `print(i)` statement is highlighted in pink, and the `i += 1` statement is highlighted in light purple.

The IPython console on the right side of the IDE shows the output of the script, displaying a sequence of numbers from 200279 to 200301, indicating that the loop is running continuously. A red circle highlights the stop button (a red square) in the top right corner of the IPython console, suggesting that the user is attempting to stop the execution of the infinite loop.

The status bar at the bottom of the IDE shows the following information: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 11, Column: 11, Memory: 68 %.

# Interrupting an infinite loop in Spyder



The screenshot shows the Spyder IDE interface. On the left, the code editor displays a Python script named `temp.py` with the following content:

```
1# -*- coding: utf-8 -*-
2"""
3Spyder Editor
4
5This is a temporary script file.
6"""
7
8i = 0
9while True:
10    print(i)
11    i += 1
```

On the right, the IPython console shows the execution of the script, which has entered an infinite loop. The console output includes:

```
File "C:\Users\rvr\anaconda3\lib\python3.7\site-packages\zmq\sugar\socket.py", line 392, in send
    return super(Socket, self).send(data, flags=flags, copy=copy, track=track)
File "zmq\backend\cython\socket.pyx", line 725, in zmq.backend.cython.socket.Socket.send
File "zmq\backend\cython\socket.pyx", line 772, in zmq.backend.cython.socket.Socket.send
File "zmq\backend\cython\socket.pyx", line 242, in zmq.backend.cython.socket._send_copy
File "zmq\backend\cython\checkrc.pxd", line 12, in zmq.backend.cython.checkrc._check_rc
KeyboardInterrupt
In [13]:
In [13]:
```

A red box with the text "click button or hit <control>C" is positioned above the console. A red arrow points from this box to a small square button in the console's title bar, which is circled in red. This button is used to interrupt the execution of the code in the console.

At the bottom of the window, the status bar shows: Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 11 Column: 11 Memory: 68 %



## Exercise

I have a list of words, and I want to combine all these words into one big word as a string.

Can I use a for-loop? If so, how would I write the loop?

Can I use a while-loop? If so, how would I write the loop?



# Functions



# Structure of a Function

```
def example(parameter):
```

Function header

```
    """ Returns: the input as a string.
```

```
    Parameter: var
```

← Specification

```
    Precondition: var is an int """
```

```
    x = str(var)
```

← Body

```
    return x
```

NB: Indentation is very important! Shows that these statements belong to that function! Indentation is needed in loops and if statements as well.





# Example of a Function

```
def greet(name):
```

```
    """This function greets to
```

```
    the person passed in as Specification
```

```
    parameter"""
```

```
    print("Hello, " + name + ". Good morning!")
```



# Functions Continued

- Functions do not always need return statements.
- Functions that behave like this are called *procedures*.

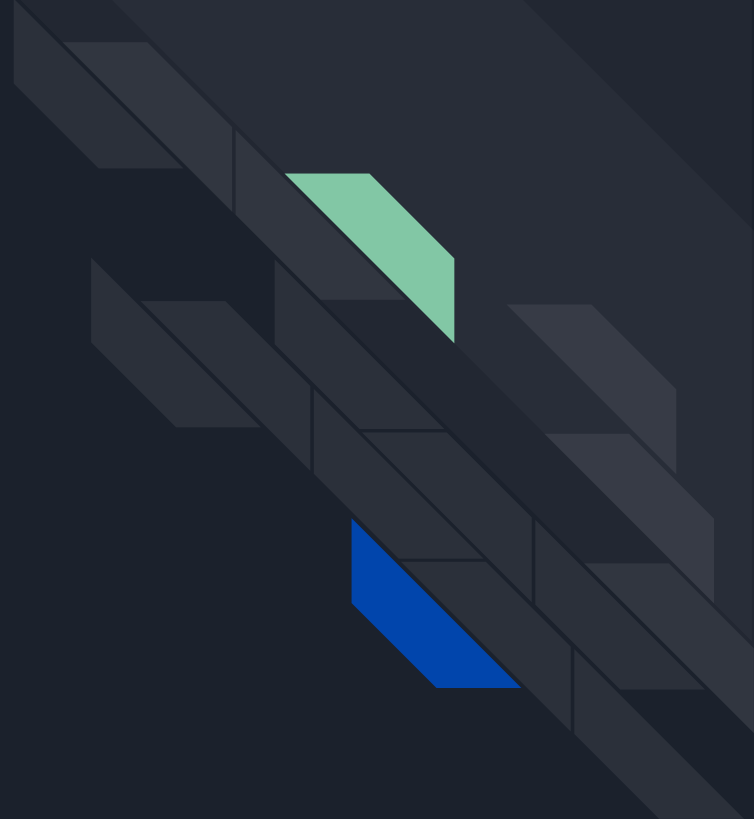


# Functions Continued

Eg. A function like `sort()` modifies the given list and does not actually return any value.

- The keyword *pass* can be used in your code as a placeholder, especially if you have not finished the function yet.
- *NB: Make sure that whatever code you are writing fulfills the specification!*

# Approaching Coding Questions





# A Three-Step Approach To Coding Problems

1. Read the question. Understand what information you have available to you and what the question wants you to do.
2. Try to write out an algorithm or solution to the problem in “English” or “pseudocode”.
3. Translate your English solution to code



# Example

Q: Write a function that takes in a number and tells you whether it is even or not.

Step 1: After reading the question, we see that we have one number as our only input. We also see that our function should either return 'even' or 'odd' or something equivalent to those outputs.

Since our function only has one input, our function header will look something like:

```
def isEven(num):
```



# Example

Q: Write a function that takes in a number and tells you whether it is even or not.

Step 2: Let's try write out a solution in English:

Given input number 'num',

- If 'num' is divisible by 2, then say it is even
- Otherwise, say it is odd



# Example

Step 3: We need to translate our English solution to code

If 'num' is divisible by 2, then say it is even

-> if num % 2 == 0:

    return "even"

Otherwise, say it is odd

-> else:

    return "odd"





# Example

So now our complete function looks like:

```
def isEven(num):  
    if num % 2 == 0:  
        return "even"  
  
    else:  
        return "odd"
```



Practice/Lab Section

Quadratic Solver

Tic Tac Toe



# Quadratic Solver Instructions

- Open the Questions folder on your computer
- Open the quadratic\_solver folder
- Open the quadratic\_solver.py file
- Read the instructions and do exactly what it says
- To test your code open the command prompt/terminal and type
  - `python tests.py`



# HACKATHON





DAY 2



# ADVANCED TOPICS



# Panel Discussions

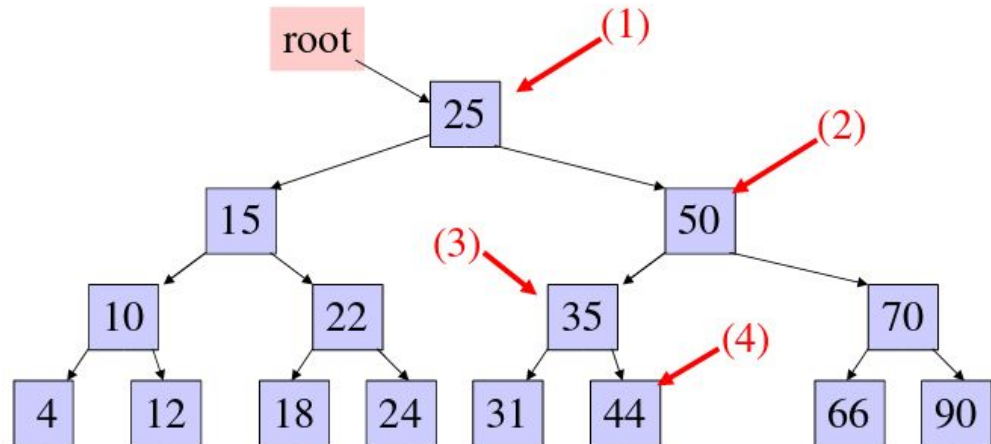


# BINARY SEARCH

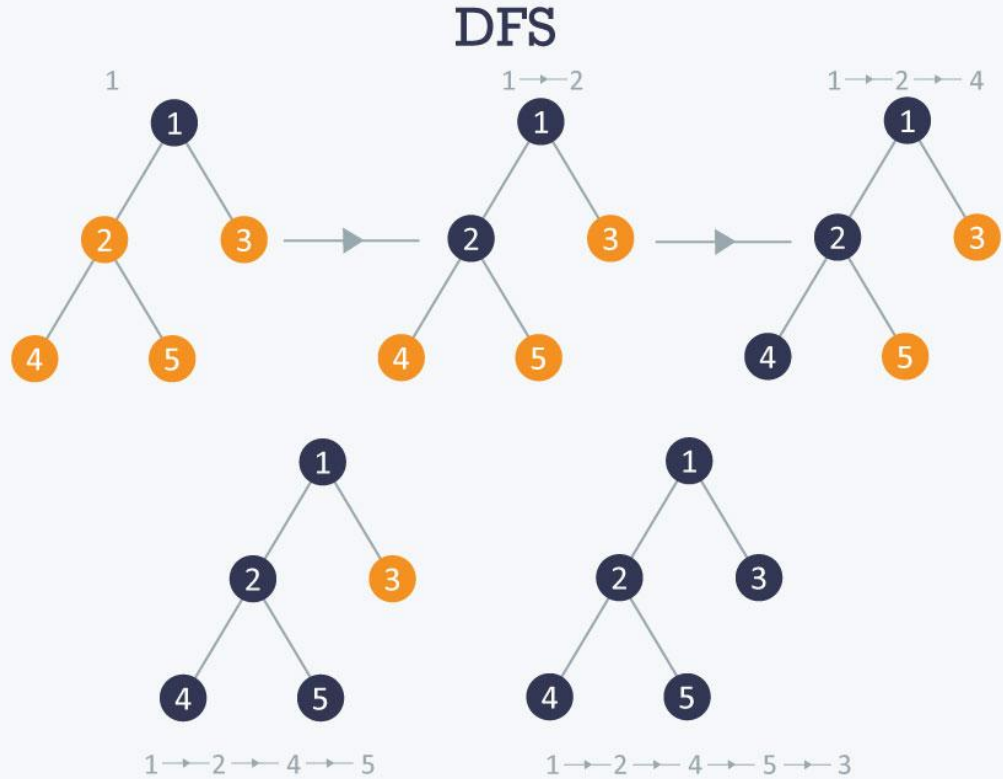
## Example: search for 45 in the tree

(key fields are show in node rather than in separate obj ref to by data field):

1. start at the root, 45 is greater than 25, search in right subtree
2. 45 is less than 50, search in 50's left subtree
3. 45 is greater than 35, search in 35's right subtree
4. 45 is greater than 44, but 44 has no right subtree so 45 is not in the BST



# DEPTH FIRST SEARCH





# OBJECT ORIENTED PROGRAMMING

```
dollar_account = BankAccount(name = "Sam", balance = "300")  
dollar_account.balance = dollar_account.balance + 500
```







# HACKATHON SESSION



# Question 1

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

```
def sum_of_multiples_of_3_or_5(n):  
    #write your code here  
    pass
```

```
def sum_of_multiples_of_3_or_5(n):  
    """If we list all the natural numbers below 10 that  
    are multiples of 3 or 5, we get 3, 5, 6 and 9.  
    The sum of these multiples is 23.  
    Find the sum of all the multiples of 3 or 5 below 1000.  
    """
```

```
#initialize a variable to store the sum  
total = 0
```

```
#Loop through the values of n from 1:n-1  
for i in range(1,n):
```

```
    #Check if i is divisible by 3 or 5  
    #NB: the % is the modulus  
    if i%3 == 0 or i%5 == 0:  
        #Add the value to the accumulator  
        total += i
```

```
#Return the sum  
return total
```







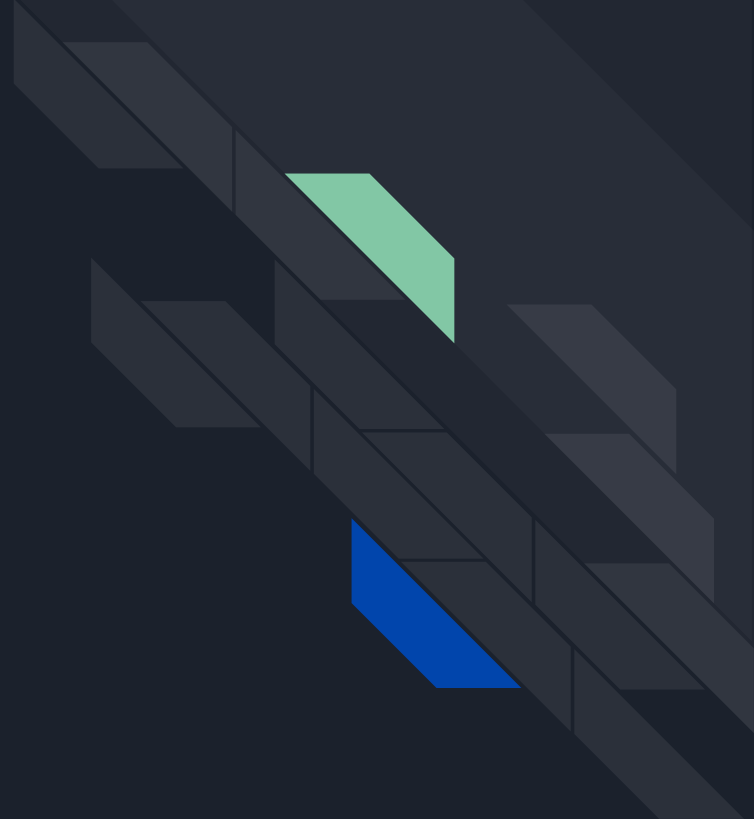
# Question 2

Write a computer program to test if a number is prime or not.

(Think about how you can make it faster)

```
def is_prime(n):  
    #write your code here  
    pass
```

```
def is_prime(n):  
    """Write a computer program to test  
    if a number is prime or not."""  
  
    #make sure all inputs are integers  
    assert isinstance(n, int)  
  
    #eliminate 0 and 1 and all negative  
    #numbers  
    if n < 2:  
        return False  
  
    #if its a prime we will find a factor #  
    between 2 and the sqaure root  
    for i in range(2, int(math.sqrt(n))+1):  
        if n%i == 0:  
            return False  
    return True
```





# Question 3

"Write a program that prints the integers between 1 and 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz"."

```
def fizz_buzz(n):  
    #write your code here  
    pass
```

```
1 def fizzbuzz(n):
2
3     assert isinstance(n,int)
4
5     for i in range(n):
6         output=""
7
8         if i%3 == 0:
9             output+="Fizz"
10
11        if i%5 == 0:
12            output+="Buzz"
13
14        if output=="":
15            output=i
16        print output
```







# Question 4

Mr. X has a vast collection of electronic music albums. Each album has an assortment of different tracks, in no particular order. Due to how large his music collection is, Mr. X has not been able to listen to all of the songs in his collection. He wants to be able to create a program that looks at an album and returns a list of the songs he has not listened to.

Mr. X was able to create a helper function 'check\_plays' that returns 1 if a song has been played before and returns 0 otherwise. He now needs to write the function 'unplayed\_songs' that takes in the album as a list of songs, and returns a list of the songs in that album that have not been listened to before. Write your implementation of 'unplayed\_songs'. You can use the function 'check\_plays' in your solution.

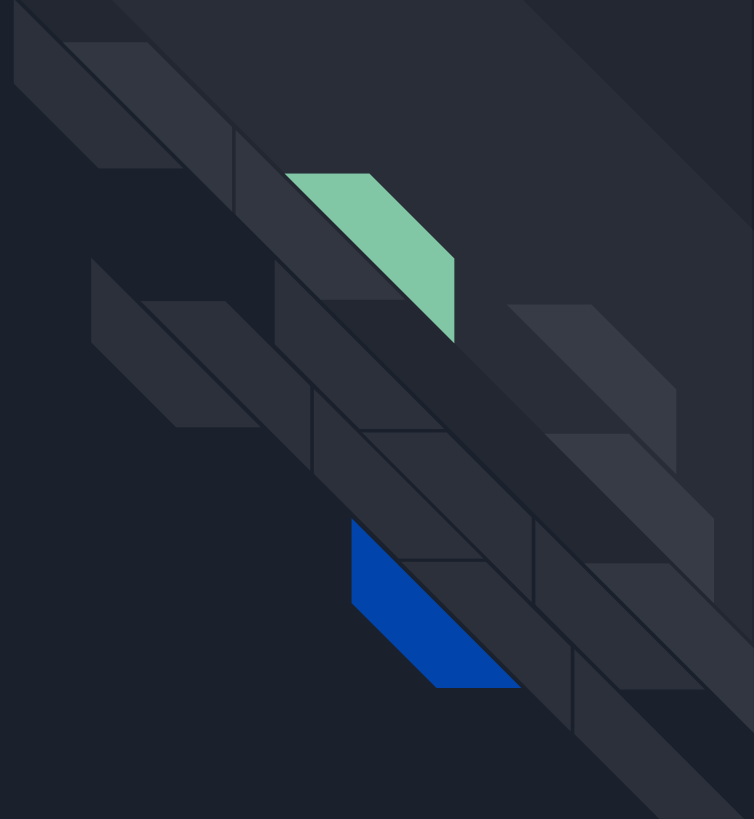
```
def unplayed_songs(album):  
    # Write your code  
    here  
    pass  
  
def check_plays(song):  
    # You should assume  
    that this function  
    works as it should.  
    pass
```

```
def unplayed_songs(album):
    # Solution
    assert isinstance(album, list)
    for x in album:
        assert isinstance(x, str)
    result = []

    # Loop through list and append
    # song to accumulator if check_plays
    # returns 0.

    for song in album:
        temp = check_plays(song)
        if temp == 0:
            result.append(song)

    return result
```





END OF SESSION

Thank You!