CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lab 8

Lab 8 is released this morning:

The goal of this lab is to write the code which takes input from the user and starts the snake game I showed you in class at the start of the year

It consists of 2 files ‘snake.py’ and ‘game,py’

snake.py can use lists and dictionaries, as parameters to game.py

We haven’t covered dictionaries so far, so either this afternoon or Wednesday we will cover the parts we need

MCQ review

Following our review of the MCQ questions last Wednesday

Q.22 was a question that was slightly confusing
and could have 2 possible answers,

So, 22 students that choose “an Error is thrown” for Q.22

will get the extra half mark ☺

MCQ review

Following our review of the MCQ questions last Wednesday

Q.22 was an question that was slightly confusing and could have 2 answers,

So, 22 students that choose “an Error is thrown” for Q.22

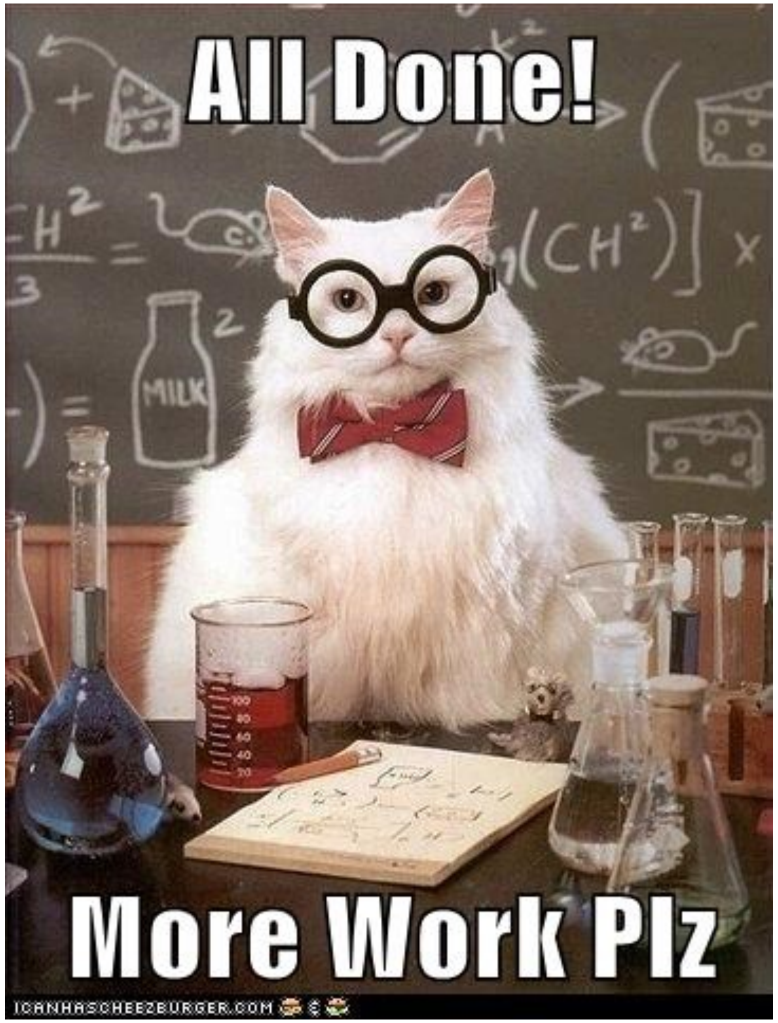
will get the extra half mark ☺

Let’s look at the remainder of the MCQ questions - Q.26

Canvas Student App

Let's Sign into this lecture now

Access Code
25684



University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

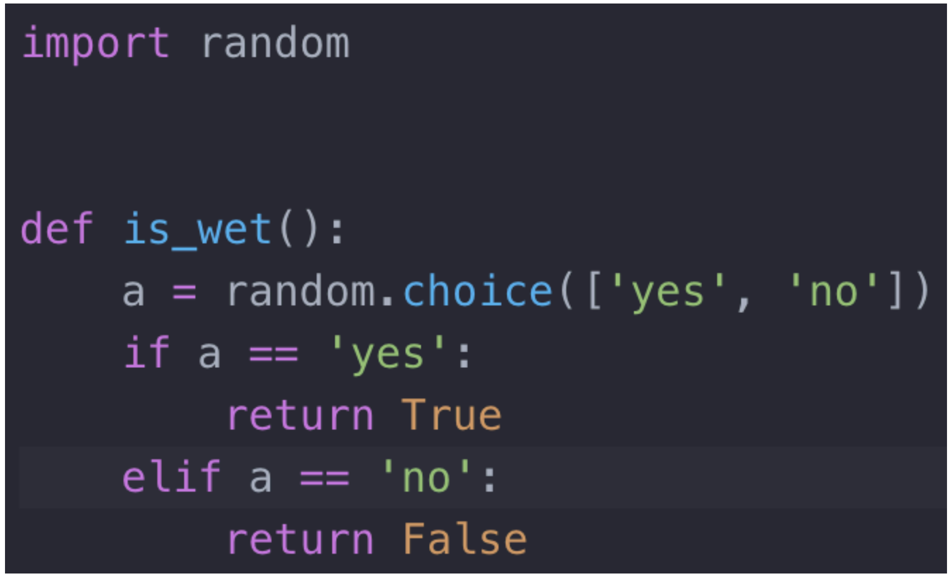
Lab Recap

Let us look at some of the issues that keep popping up in the lab submissions:

Let’s start with Boolean Functions:

Lab Recap

This is a Boolean functions from Lab 7, called is_wet()



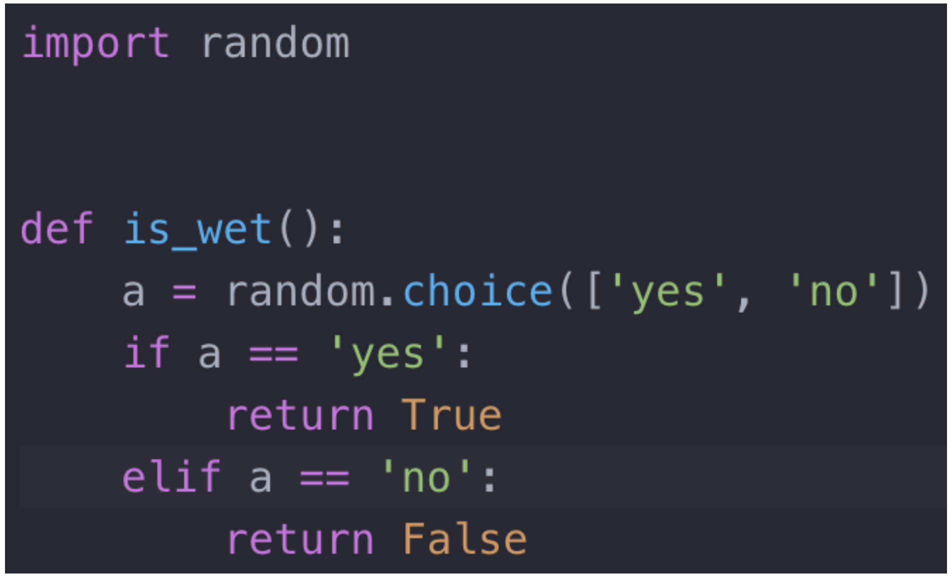
```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

The instruction in the Lab was that the function is called is_wet() defined without a parameter and returns True or False

So the Code is good ☺

Lab Recap

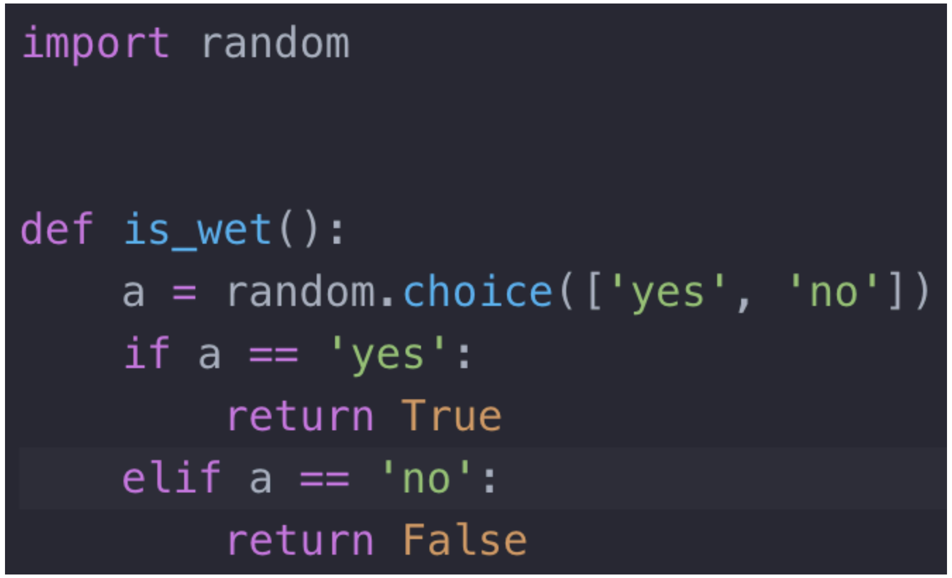
But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Lab Recap

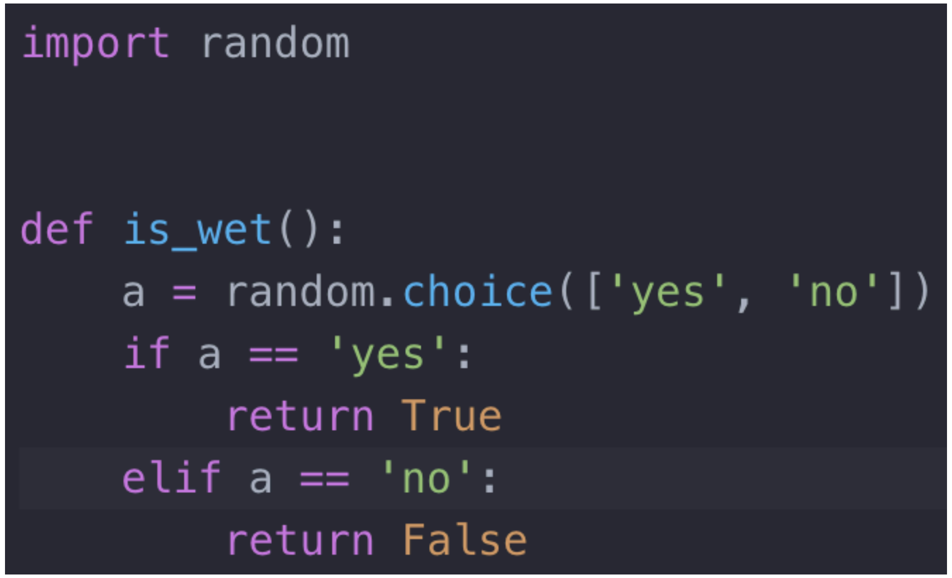
But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Lab Recap

But the code can be refined, so let’s start with the returned choice

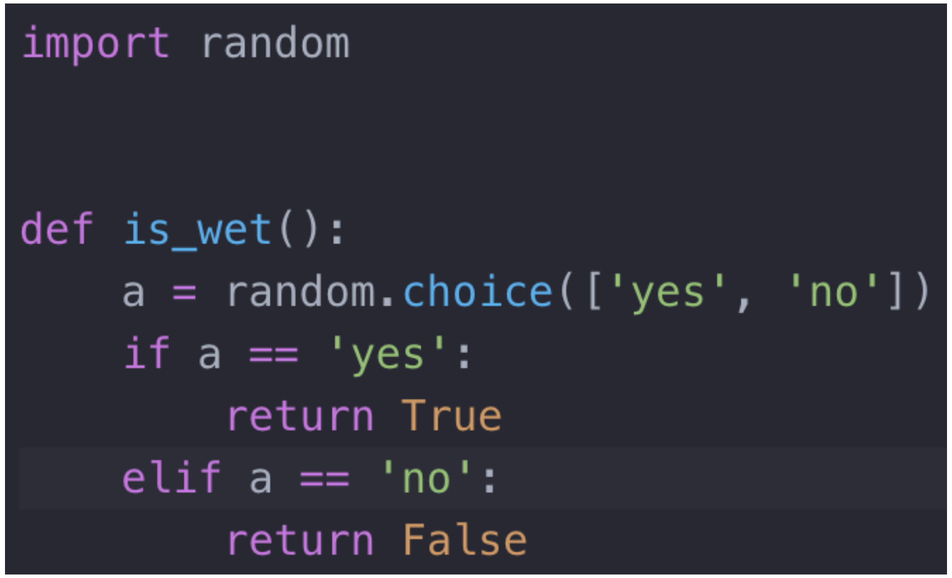


```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?

Lab Recap

But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

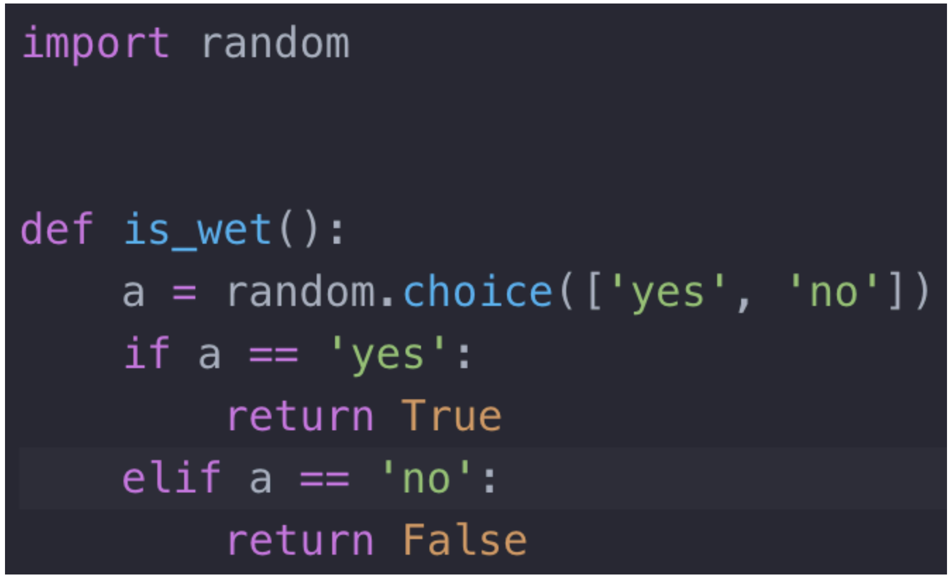
Do we need to recheck if ‘a’ is no ?

No, ‘a’ can only be one of two choices.

So we only need to check for one of them

Lab Recap

But the code can be refined, so let’s start with the returned choice

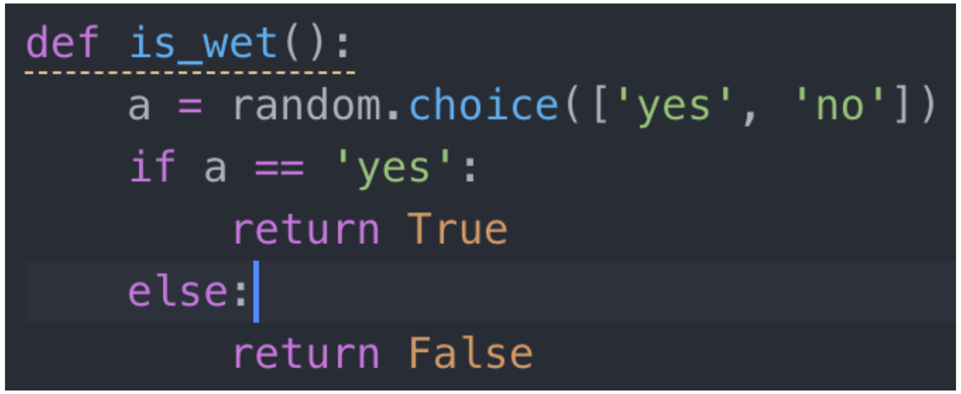


```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?

No, ‘a’ can only be one of two choices.

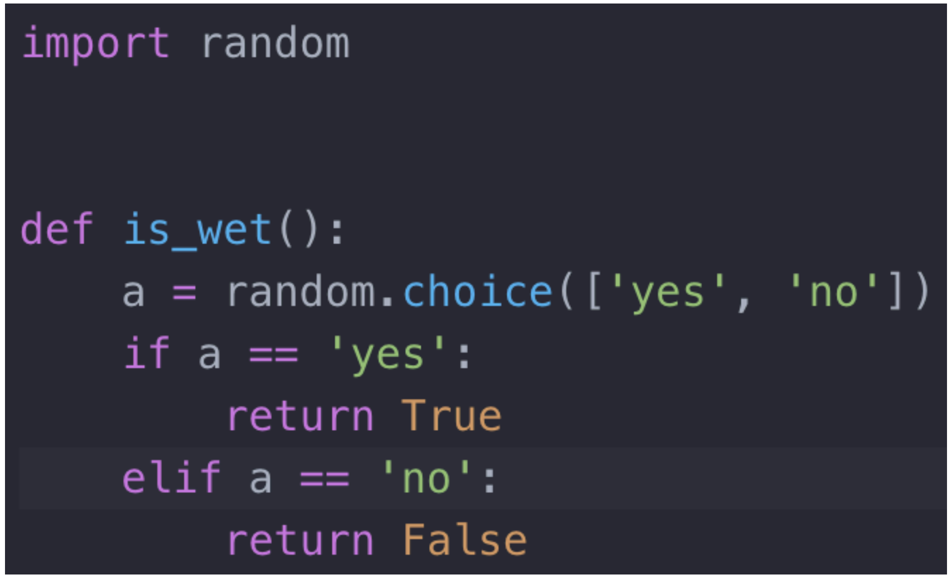
So we only need to check for one of them



```python
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    else:
        return False
```

Lab Recap

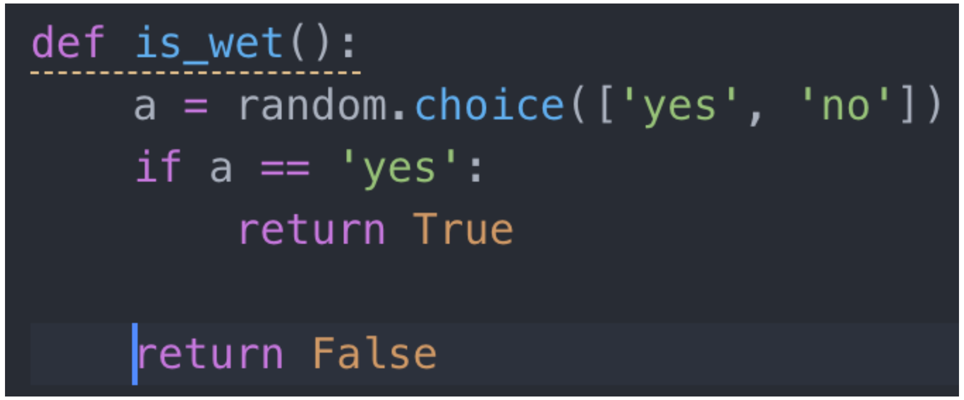
But the code can be refined, so let's start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

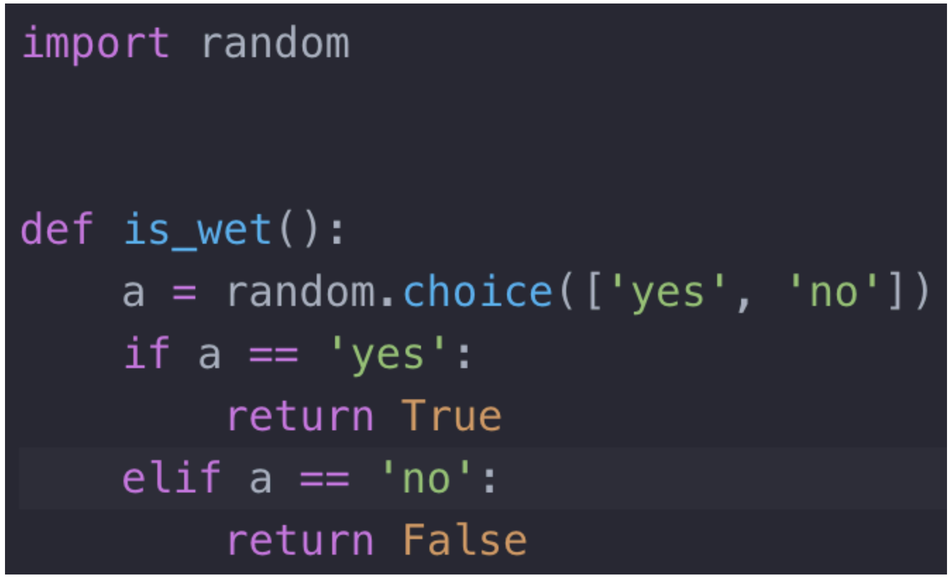
We can actually remove the else.



```python
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

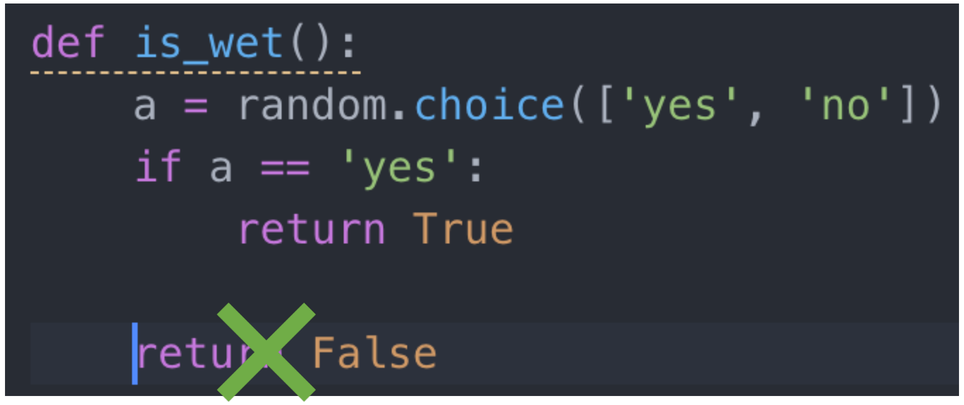
Lab Recap

But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?



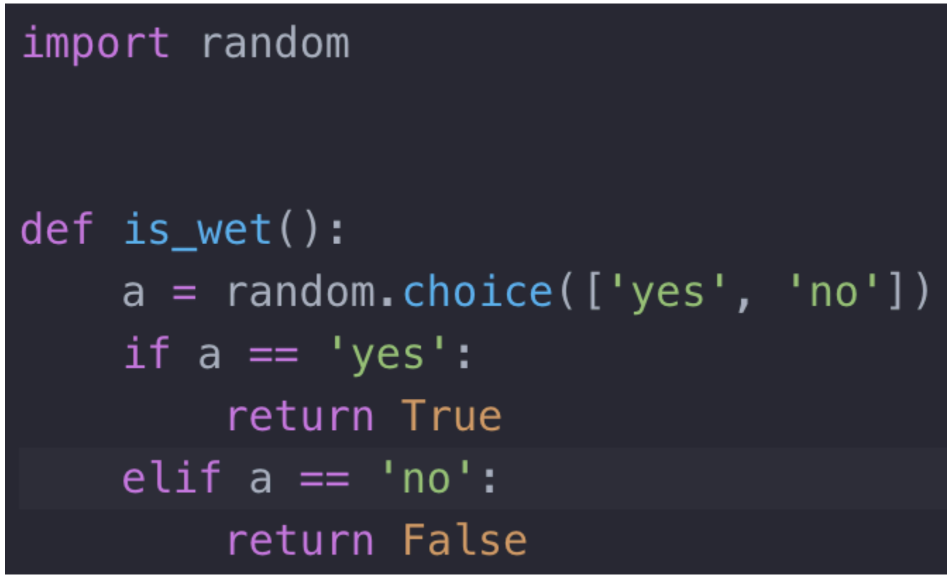
```python
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

We can actually remove the else.

If a is ‘yes’, then True is returned

Lab Recap

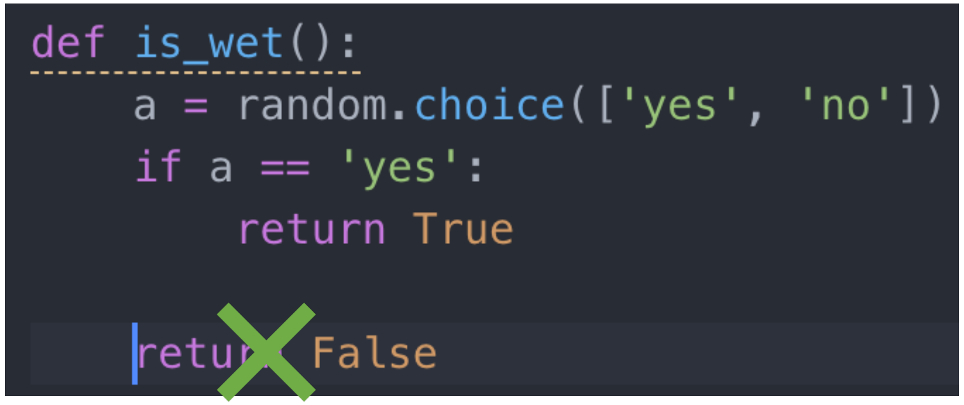
But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?

This works for return, not for print(), why??



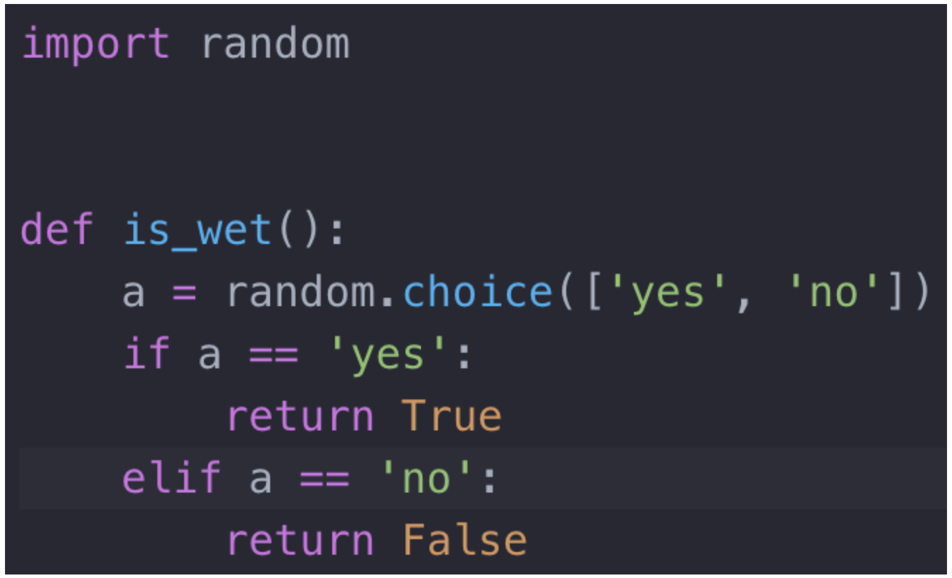
```python
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

We can actually remove the else.

If a is ‘yes’, then True is returned

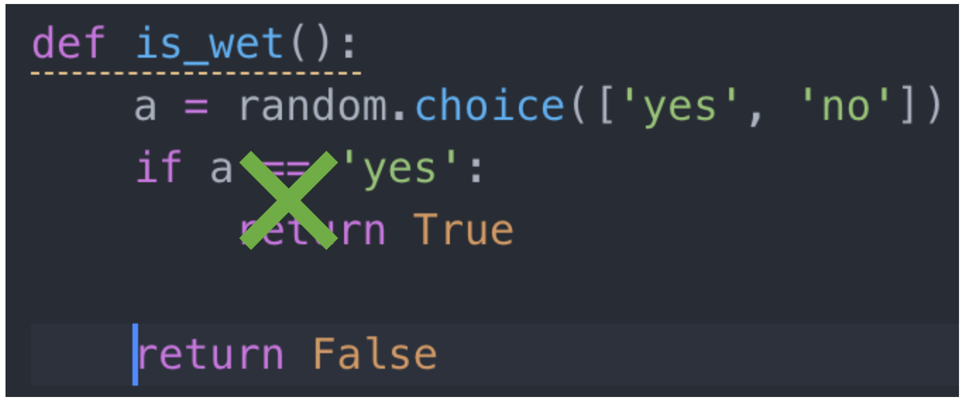
Lab Recap

But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?



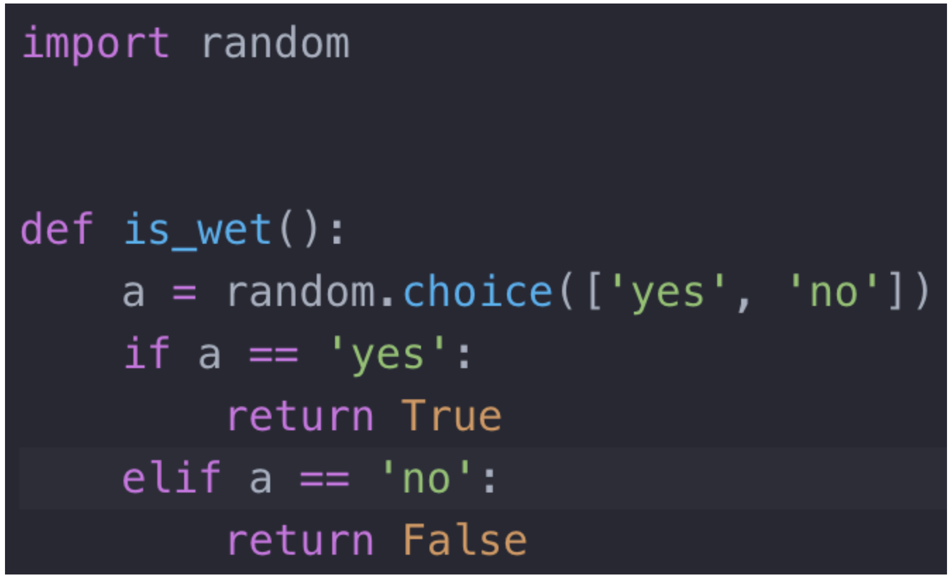
```python
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

We can actually remove the else.

If a is ‘no’, then False is returned

Lab Recap

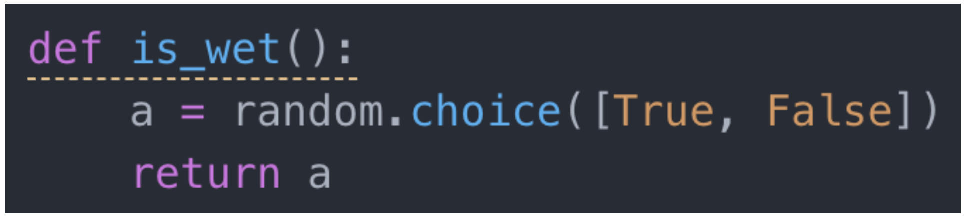
But the code can be refined, so let's start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

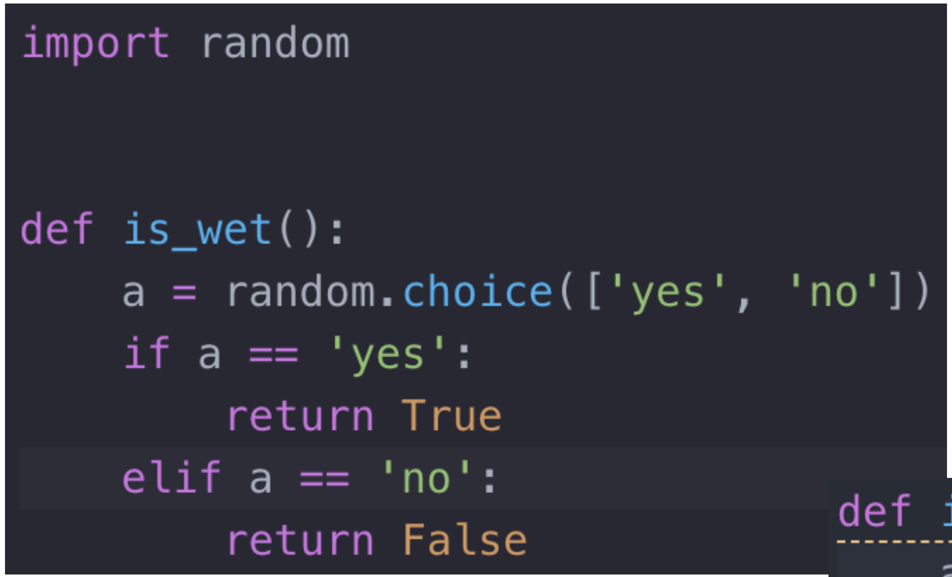
Actually, we can remove the 'if', if we set our choice to True or False



```python
def is_wet():
    a = random.choice([True, False])
    return a
```

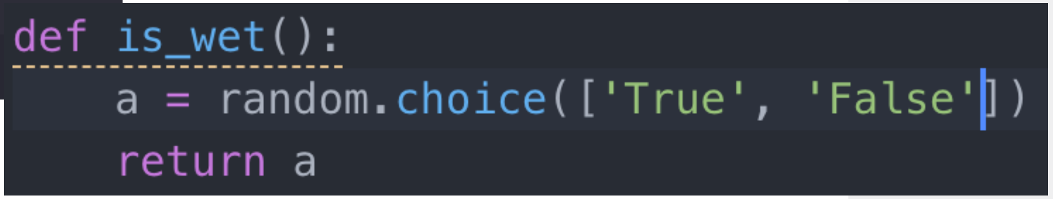
Lab Recap

But the code can be refined, so let’s start with the returned choice



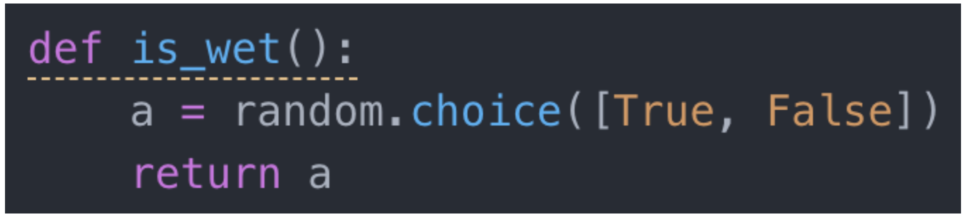
```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?



```python
def is_wet():
    a = random.choice(['True', 'False'])
    return a
```

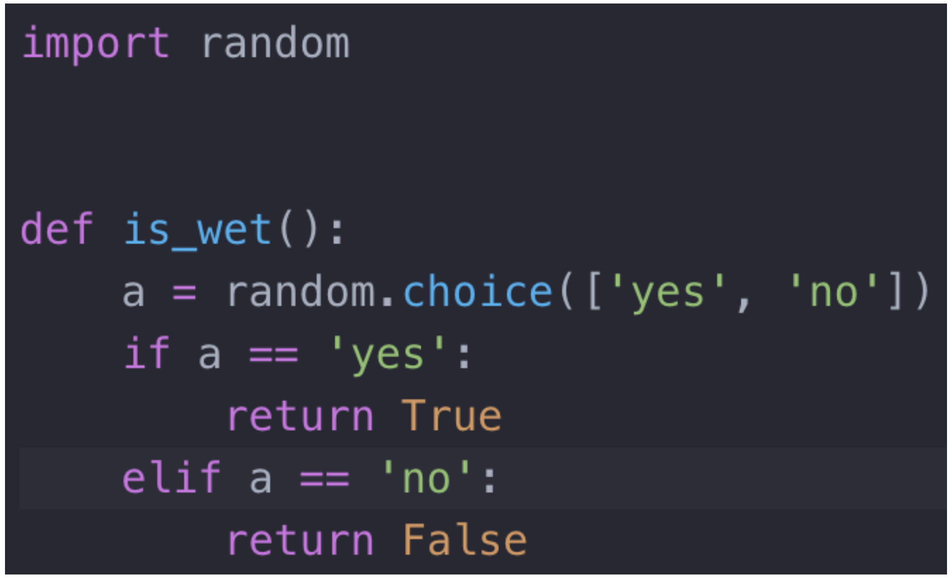
Note: this is not the same, these are strings with values ‘True’ and ’False’



```python
def is_wet():
    a = random.choice([True, False])
    return a
```

Lab Recap

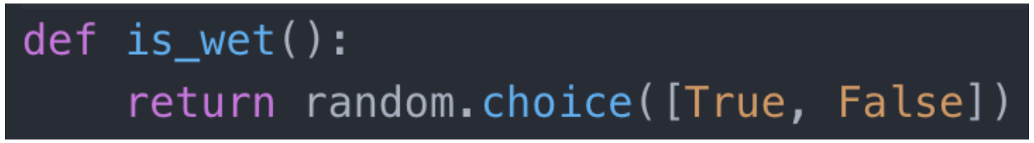
But the code can be refined, so let’s start with the returned choice



```python
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if ‘a’ is no ?

Actually, we can remove the ‘a’ variable, if we just return the random choice

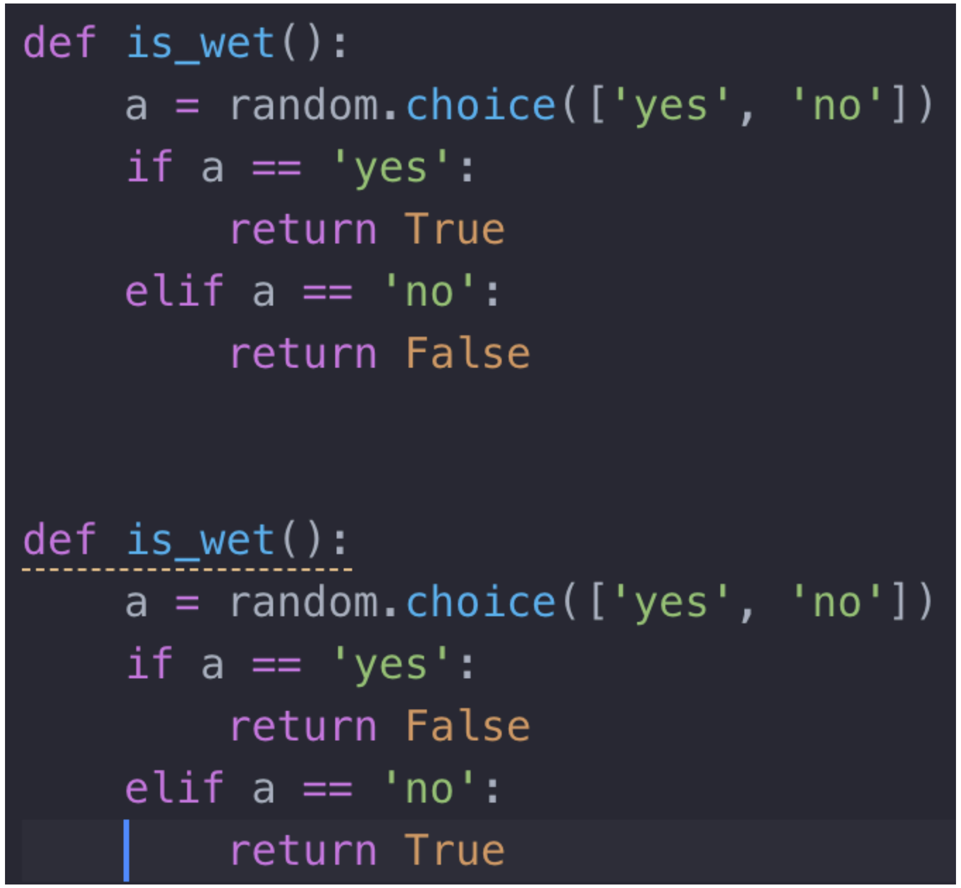


```python
def is_wet():
    return random.choice([True, False])
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lab Recap

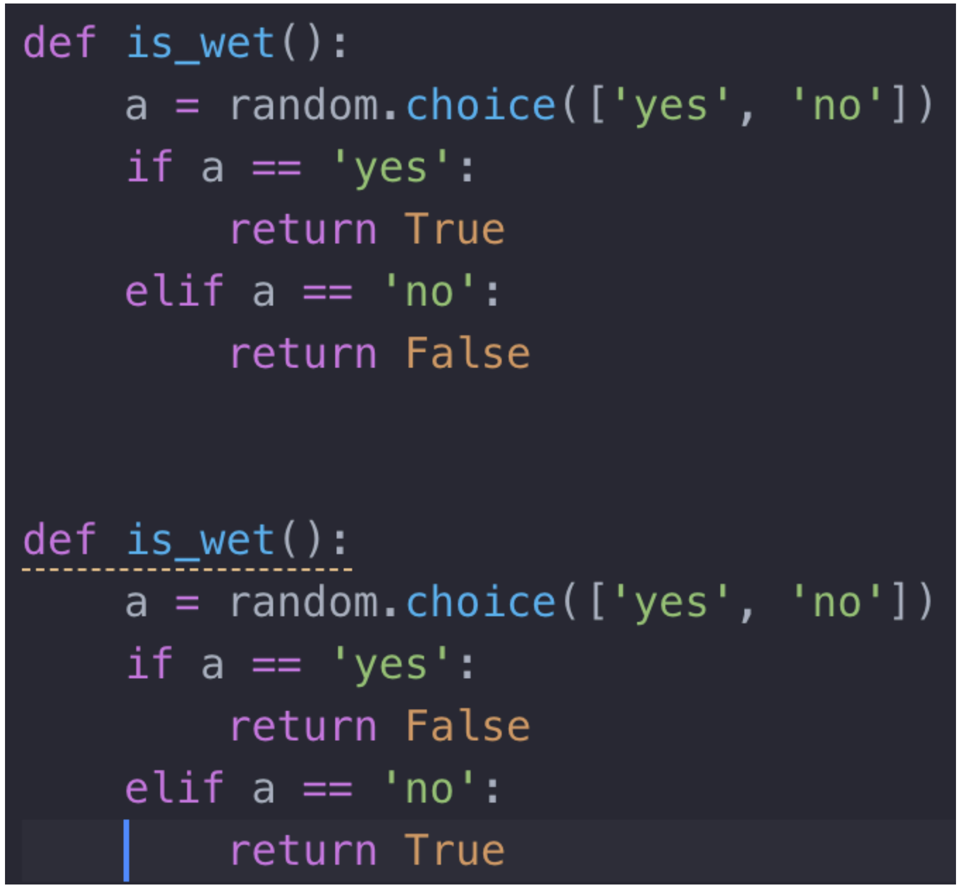
Before I move away from this function, lets look at functions with the same name



```python
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Lab Recap

Before I move away from this function, lets look at functions with the same name

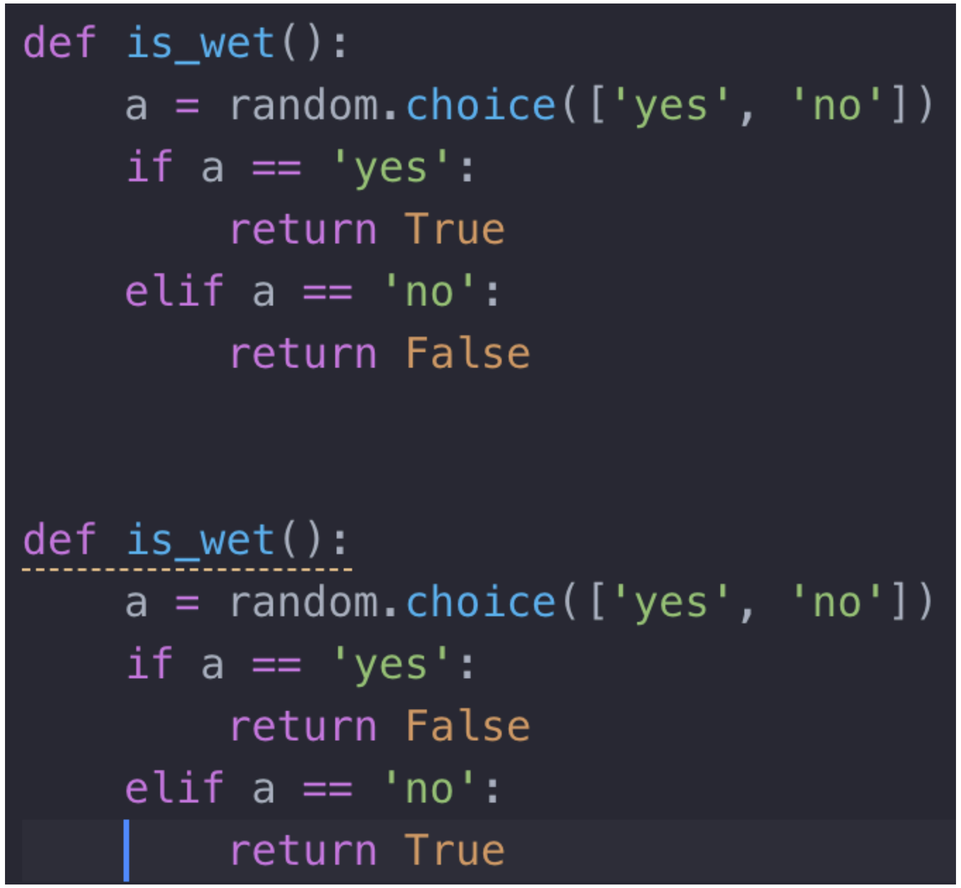


```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Note in the second definition, I reversed the returned values

Lab Recap

Before I move away from this function, lets look at functions with the same name



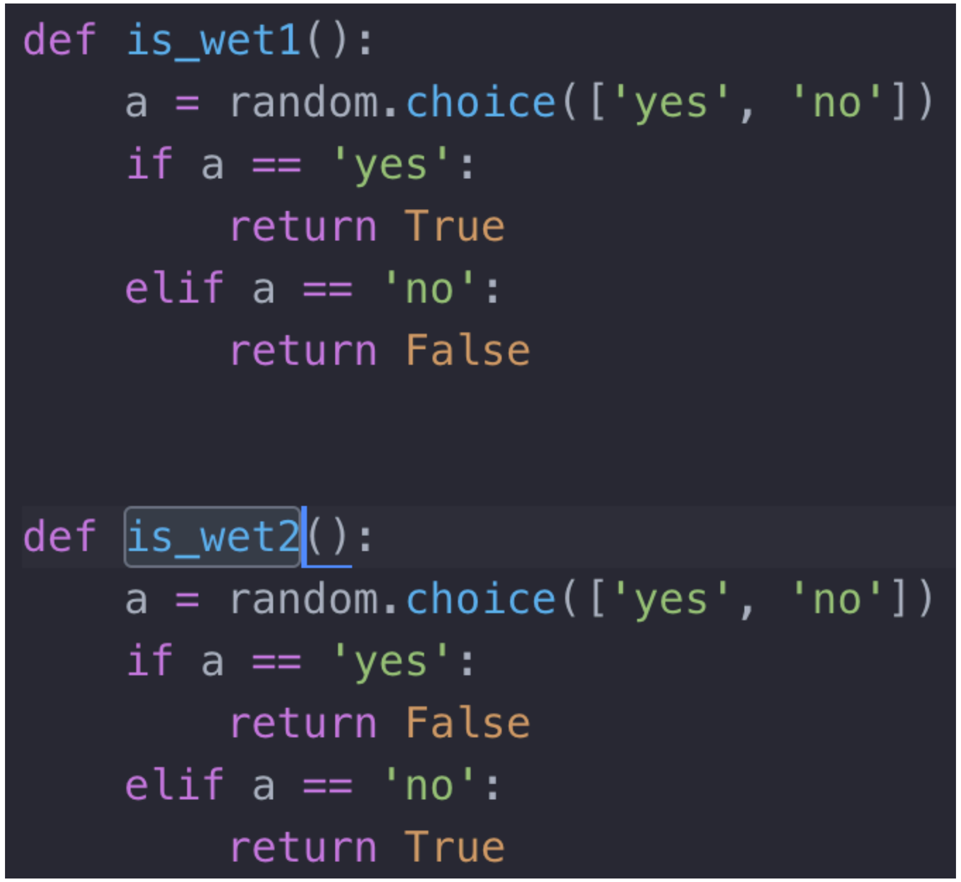
```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Note in the second definition, I reversed the returned values

Python will only remember the last time a function, or a variable, was defined, so here ’yes’ will always return False

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

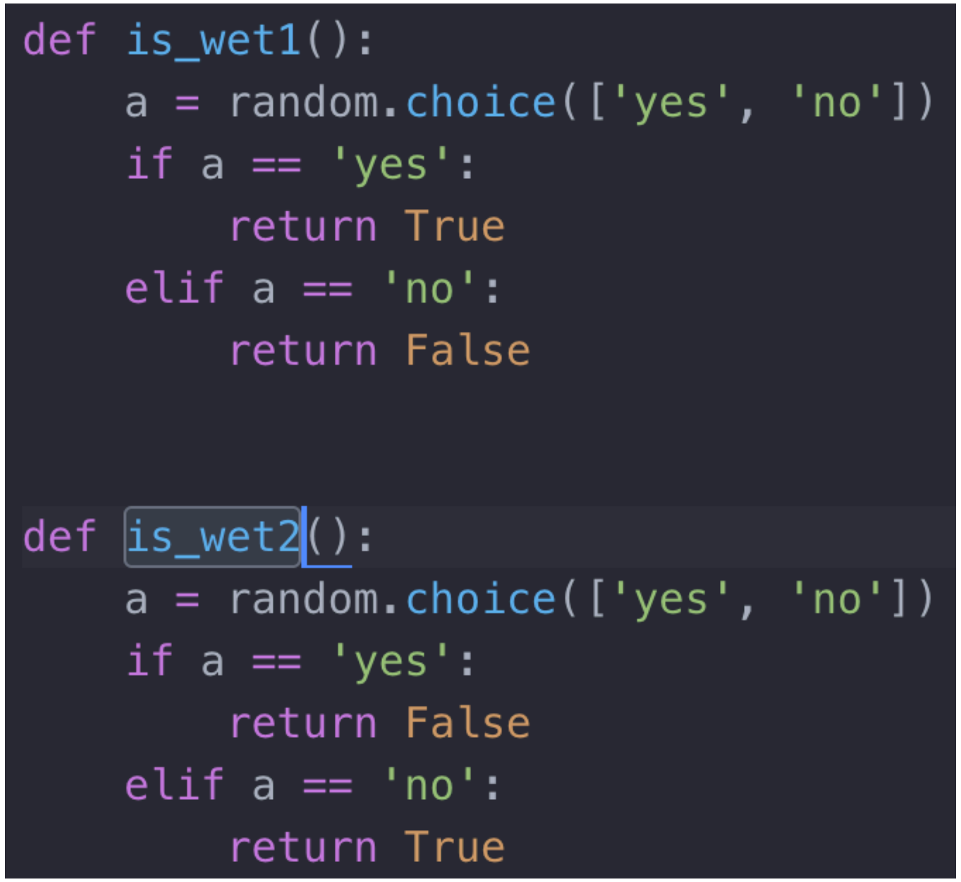


```python
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

Remember variables with the same text but with different upper and lower are different

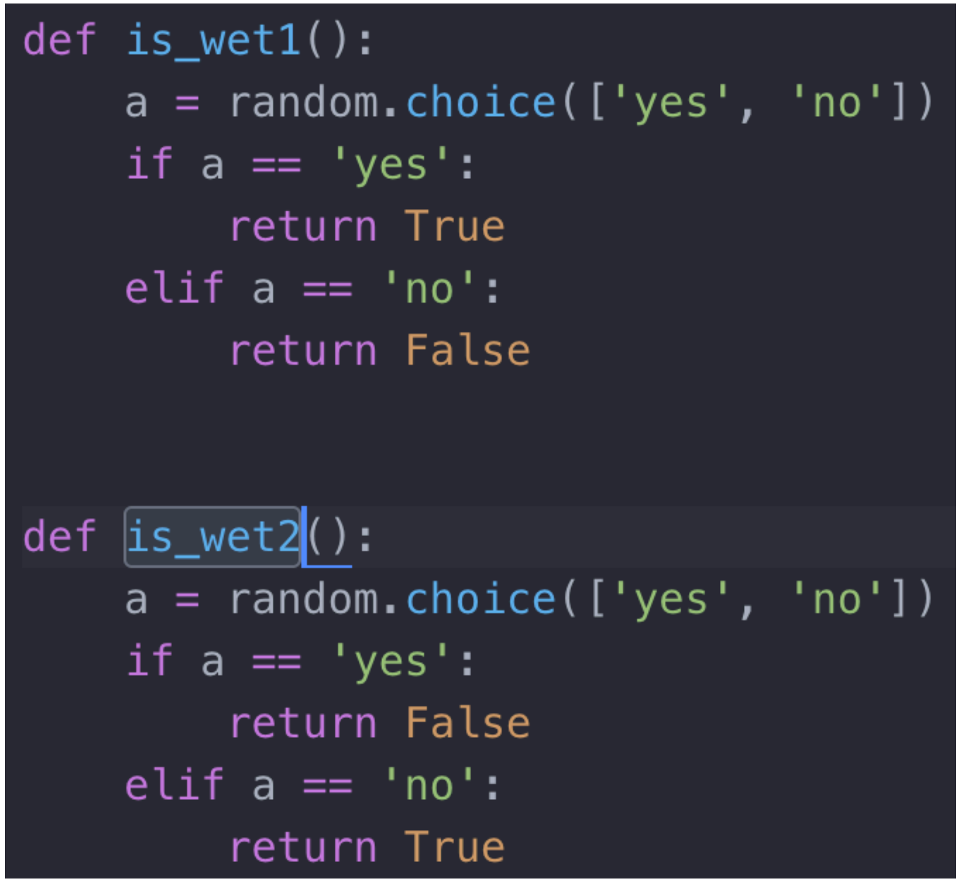


```
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

Remember variables with the same text but with different upper and lower are different



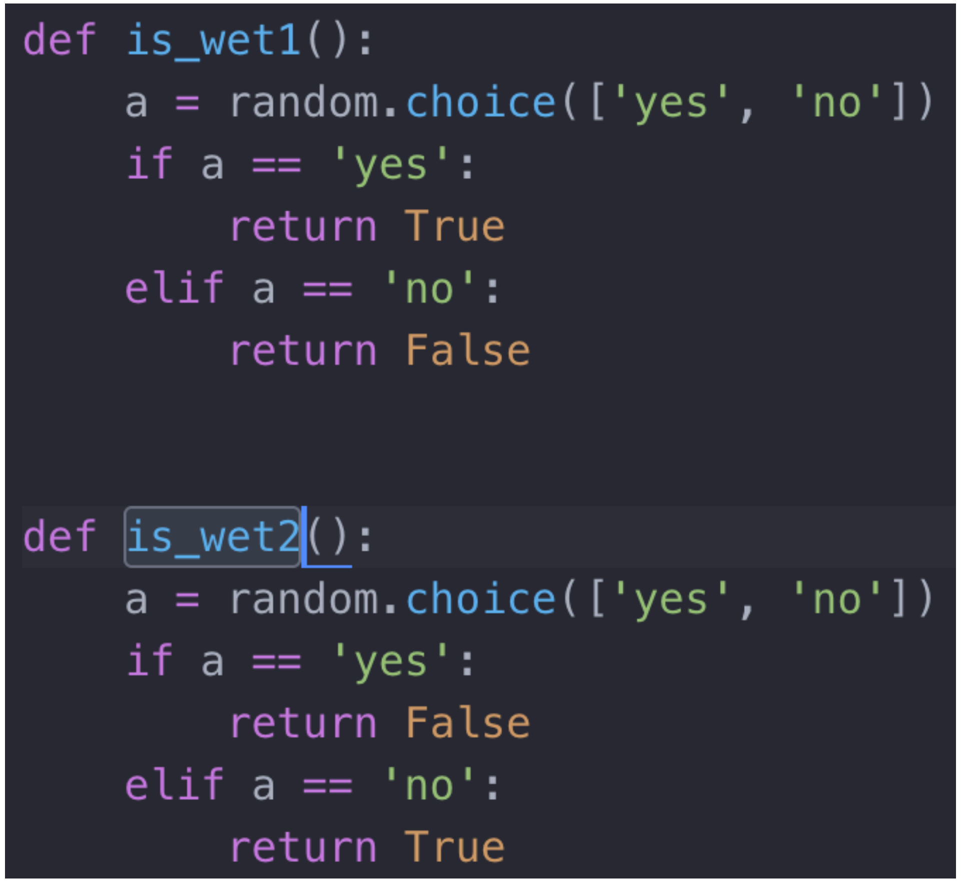
```
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

So are functions with slightly different names, they are also different

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

Remember variables with the same text but with different upper and lower are different



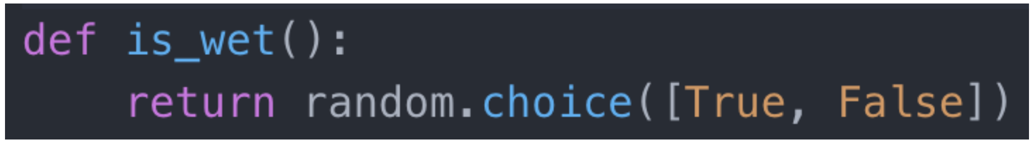
```python
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

So are functions with slightly different names, they are also different

In Android phones, the onCreate() function is called to launch the app (ish), so if you wrote onCreate1() it will not work

Lab Recap

Now, let’s return to our revised is_wet() Boolean Function



```python
def is_wet():
    return random.choice([True, False])
```

This will return True or False

So how do we check for this?

Using if and while?

Let’s see….

Lab Recap

Using if and while?

```
def is_wet():
    return random.choice([True, False])
```

```
if is_wet() == True:
    print("OMG - mom is not going to be happy....")

while is_wet() == True:
    print("OMG - mom is really not going to be happy....")
```

Lab Recap

Using if and while?

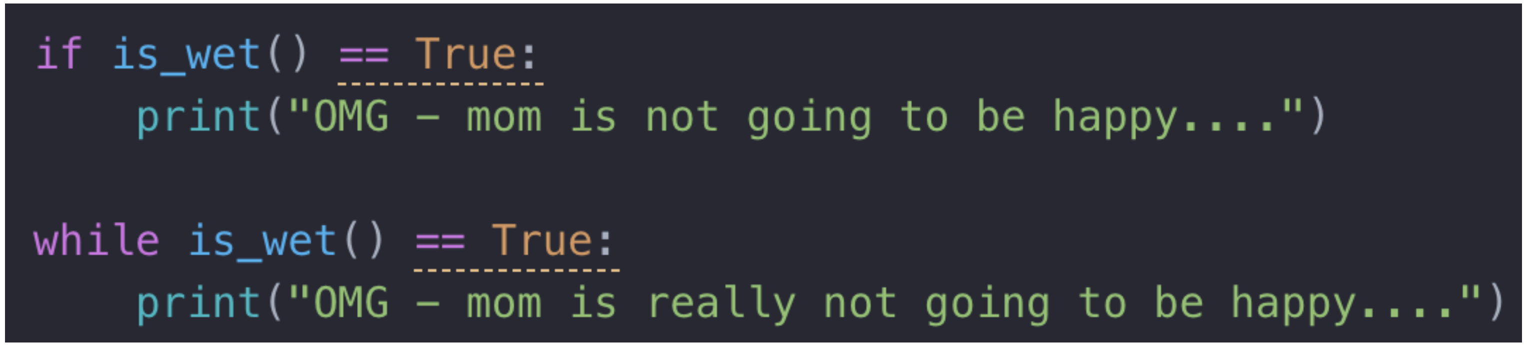
```
def is_wet():
    return random.choice([True, False])
```

```
if is_wet() == True:
    print("OMG - mom is not going to be happy....")

while is_wet() == True:
    print("OMG - mom is really not going to be happy....")
```

```
# output
# OMG - mom is not going to be happy....
# OMG - mom is really not going to be happy....
# OMG - mom is really not going to be happy....
# OMG - mom is really not going to be happy....
# OMG - mom is really not going to be happy....
```

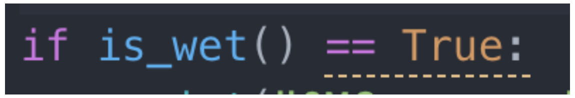
Lab Recap

Let’s look at the if condition statement



```
if is_wet() == True:
    print("OMG - mom is not going to be happy....")

while is_wet() == True:
    print("OMG - mom is really not going to be happy....")
```

We are checking if the returned value from is_wet() is True



```
if is_wet() == True:
```

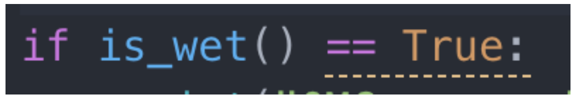
I see where you are coming from,

but let’s look at this a little bit more

Lab Recap

First off, see the yellow underline…

Python is not happy with this code and issues a warning:



```python
if is_wet() == True:
```

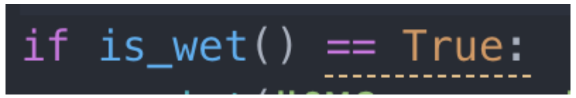
E712 comparison to True should be

'if cond is True:' or 'if cond:'

Lab Recap

First off, see the yellow underline…

Python is not happy with this code and issues a warning:

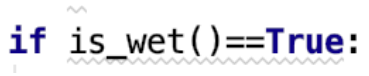


```python
if is_wet() == True:
```

E712 comparison to True should be

'if cond is True:' or 'if cond:'

It is the same in PyCharm – white underline:

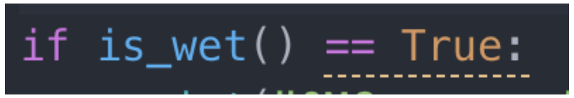


```python
if is_wet()==True:
```

Expression can be simplified

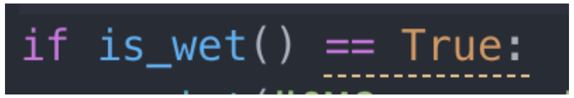
Lab Recap

So, let’s run the code and see what happens…



```
if is_wet() == True:
```

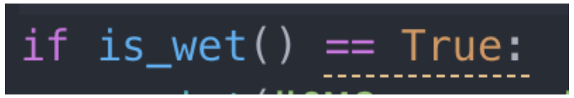
Let’s say is_wet() returns True:



```
if is_wet() == True:
```

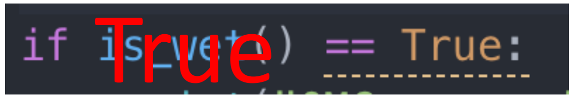
Lab Recap

So, let's run the code and see what happens…



```
if is_wet() == True:
```

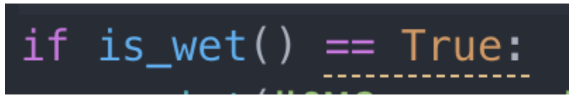
Let's say is_wet() returns True:



```
if is_wet() == True:
```

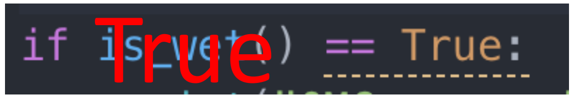
Lab Recap

So, let's run the code and see what happens…



```
if is_wet() == True:
```

Let's say is_wet() returns True:

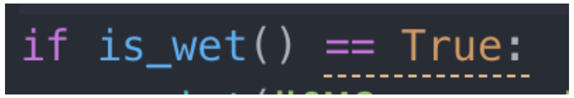


```
if is_wet() == True:
```

Now we are checking if True == True

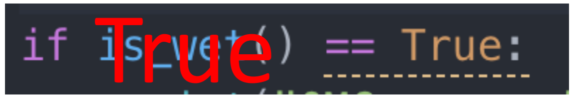
Lab Recap

So, let’s run the code and see what happens…



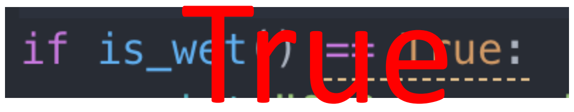
```
if is_wet() == True:
```

Let’s say is_wet() returns True:



```
if is_wet() == True:
```

Now we are checking if True == True

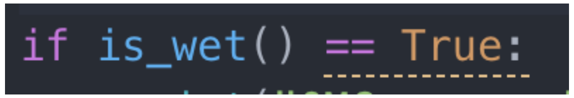


```
if is_wet() == True:
```

So all good, right?? Well…

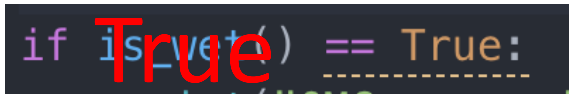
Lab Recap

So, let's run the code and see what happens…



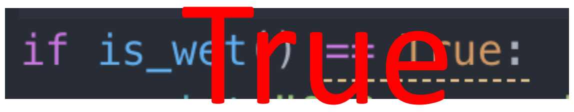
```
if is_wet() == True:
```

Let's say is_wet() returns True:



```
if is_wet() == True:
```

Now we are checking if True == True



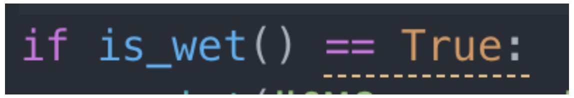
```
if is_wet() == True:
```

So all good, right?? Well…

Do you see that the value of is_wet() is actually the value we use?

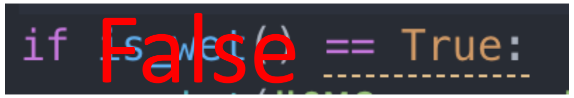
Lab Recap

Let’s see what happens when is_wet() is False:



```
if is_wet() == True:
```

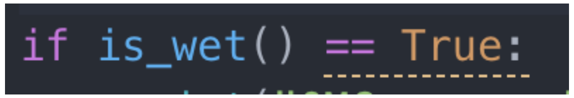
is_wet() returns False:



```
if False == True:
```

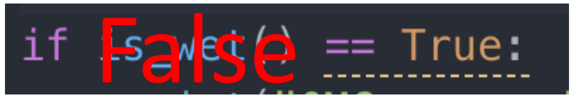
Lab Recap

Let's see what happens when is_wet() is False:



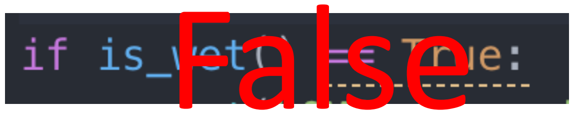
```
if is_wet() == True:
```

is_wet() returns False:



```
if False == True:
```

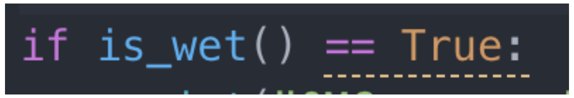
Now we are checking if False == True



```
if False:
```

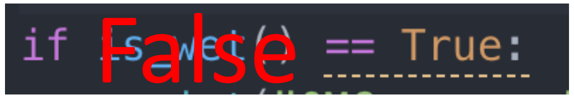
Lab Recap

Let’s see what happens when is_wet() is False:



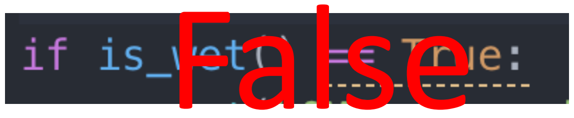
```
if is_wet() == True:
```

is_wet() returns False:



```
if is_wet() == True:
```

Now we are checking if False == True

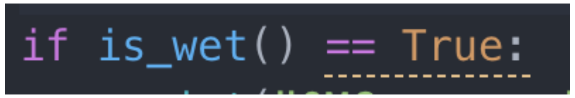


```
if is_wet() == True:
```

Again, do you see that the value of is_wet() is actually the value we use?

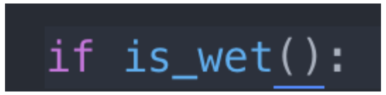
Lab Recap

So, when we have a conditional statement, using a Boolean function, we just need to call the function:



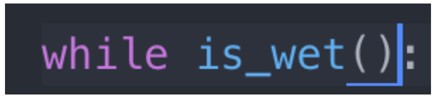
```
if is_wet() == True:
```

Becomes:



```
if is_wet():
```

or

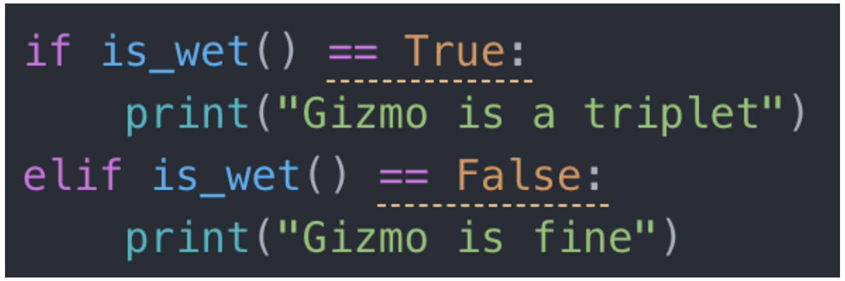


```
while is_wet():
```

Lab Recap

One last item on is_wet()

I am also seeing this kind of code:



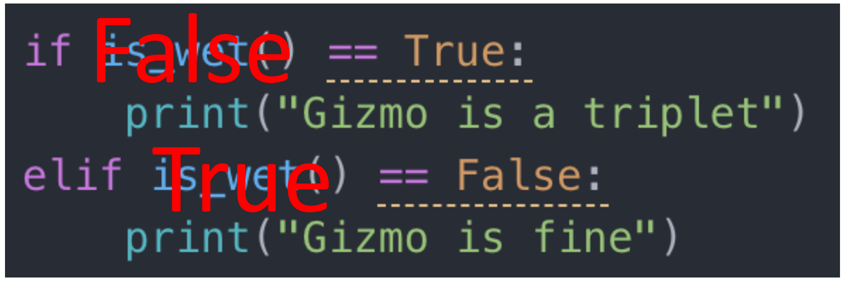
```python
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Here we have two calls to is_wet() and if we assume each returns a different value, we could get:

Lab Recap

One last item on is_wet()

I am also seeing this kind of code:



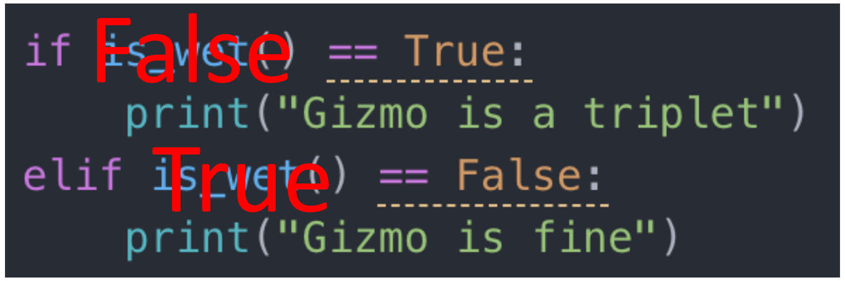
```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Here we have two calls to is_wet() and if we assume each returns a different value, we could get:

Lab Recap

One last item on is_wet()

I am also seeing this kind of code:



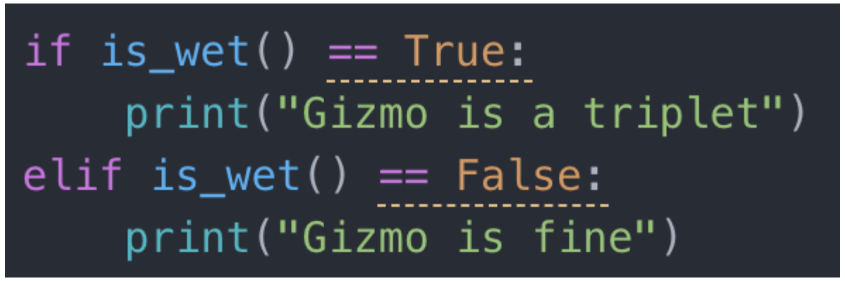
```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Here we have two calls to is_wet() and if we assume each returns a different value, we could get:

Nothing being printed…

Lab Recap

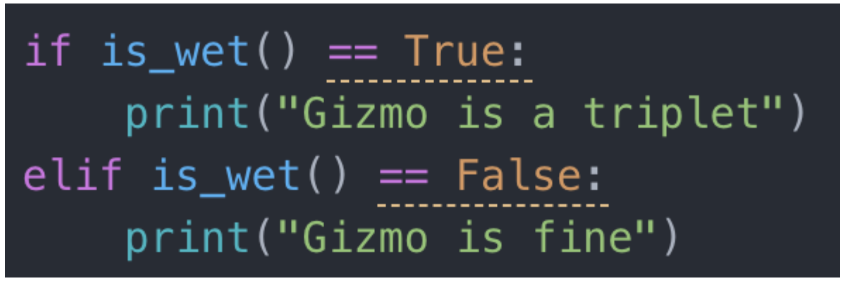
Take a moment and think how this should be rewritten:



```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Lab Recap

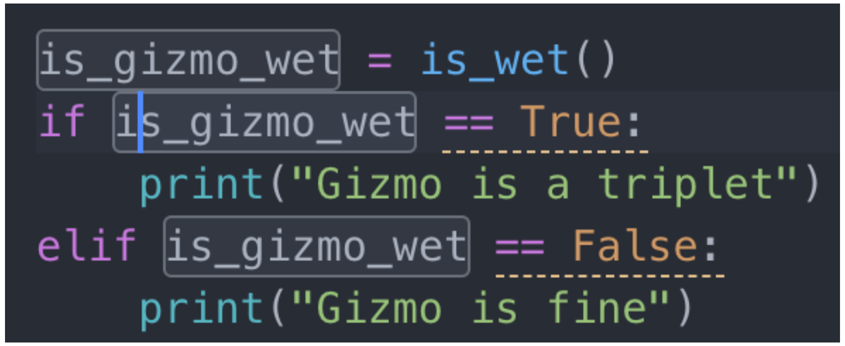
Take a moment and think how this should be rewritten:



```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

If we take is_wet() and save in a variable,

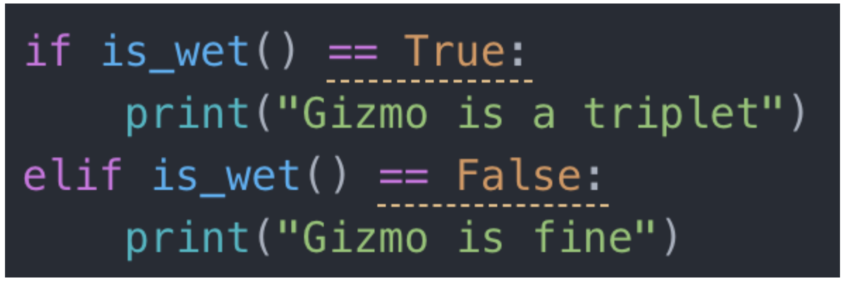
at least we only call is_wet() once



```
is_gizmo_wet = is_wet()
if is_gizmo_wet == True:
    print("Gizmo is a triplet")
elif is_gizmo_wet == False:
    print("Gizmo is fine")
```

Lab Recap

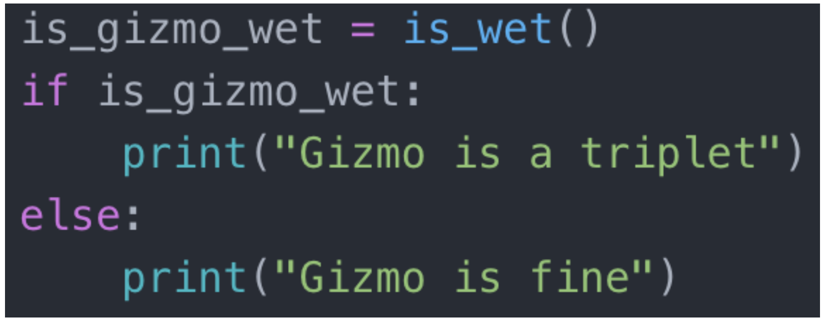
Take a moment and think how this should be rewritten:



```python
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Actually once we define the variable, it is a Boolean

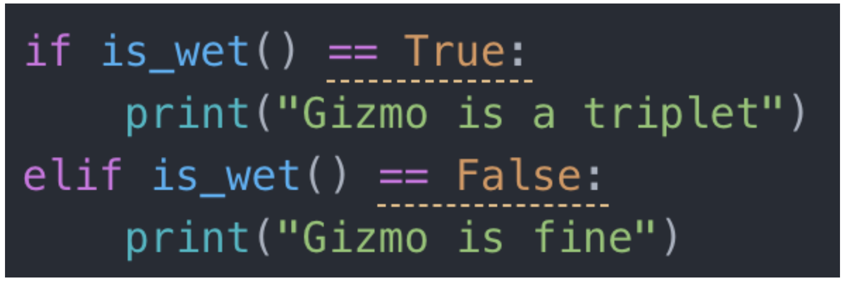
So we can use the variable as our conditional statement



```python
is_gizmo_wet = is_wet()
if is_gizmo_wet:
    print("Gizmo is a triplet")
else:
    print("Gizmo is fine")
```

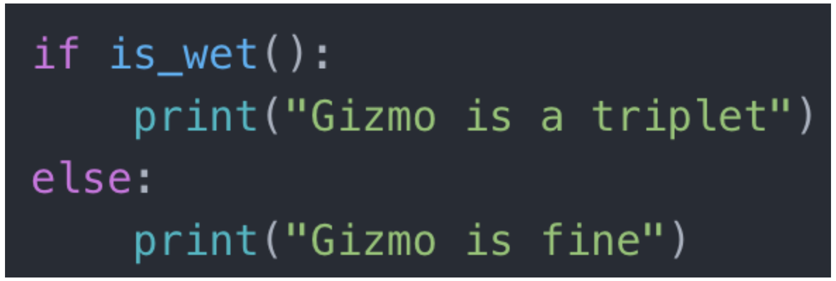
Lab Recap

Take a moment and think how this should be rewritten:



```python
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

But the best thing to do, is just use is_wet()



```python
if is_wet():
    print("Gizmo is a triplet")
else:
    print("Gizmo is fine")
```

One call, no additional variables, and easy to follow

Lab Recap

Minor items:

Don’t forget to include a docString with every function you write

Include the name of the file, your name and student number at the top of every file you write

Call your functions from the main.py file, not in funcitons.py

At a minimum, test all the examples I give you in the assignment sheet, not just one of them

Lab Recap

Minor items:

Do not change what I am asking for, if I state that stairs() needs a minimum of 2 steps, then you code for that. Even if a stairs could be defined as a single step…

Always define the function and add a return

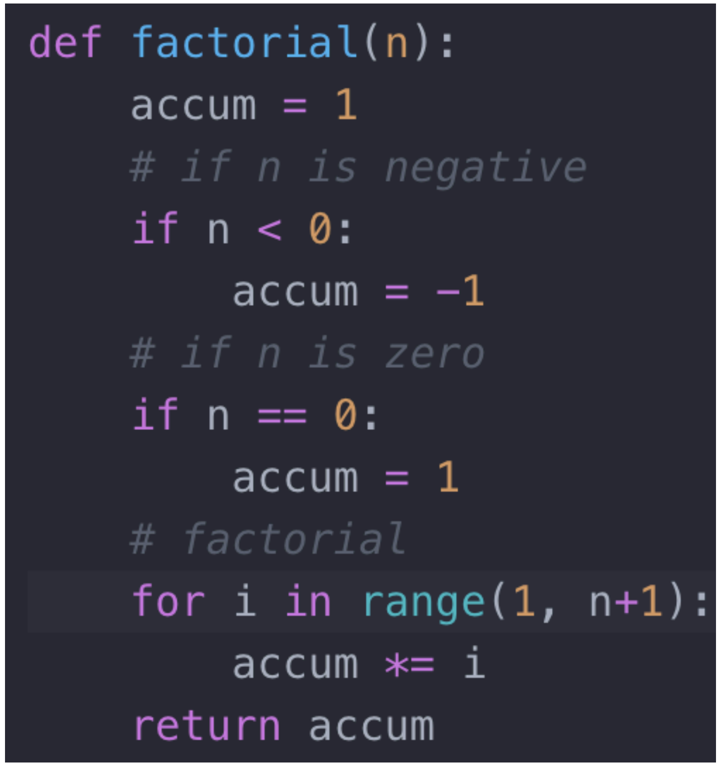
Even if you never add any code to the function

```
def function(parameters):
    return
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lab Recap

Let’s move to factorial(n):

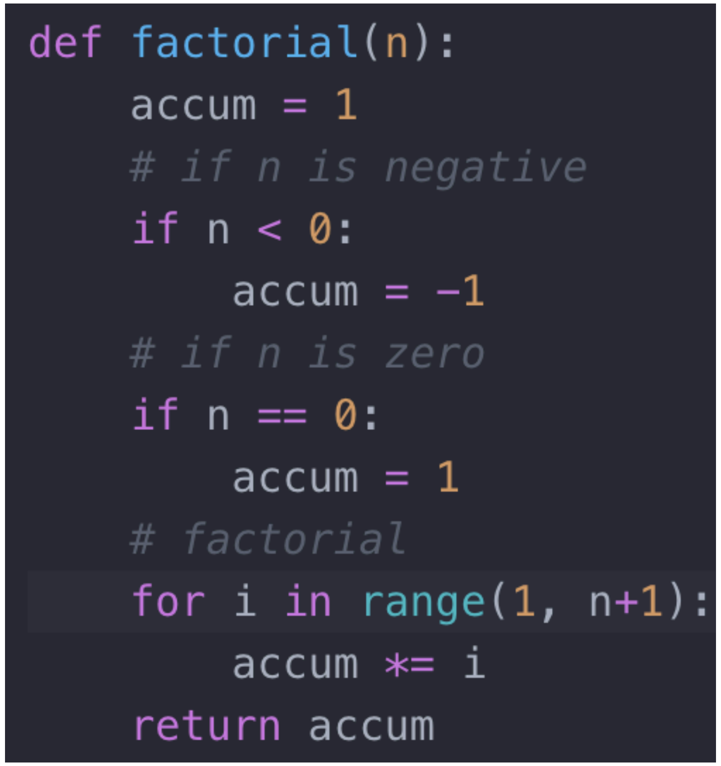


```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

factorial(n) takes in a number and returns the factorial (n*n-1*n-2*…2*1)

Lab Recap

Let’s move to factorial(n):



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

factorial(n) takes in a number and returns the factorial (n*n-1*n-2*…2*1)

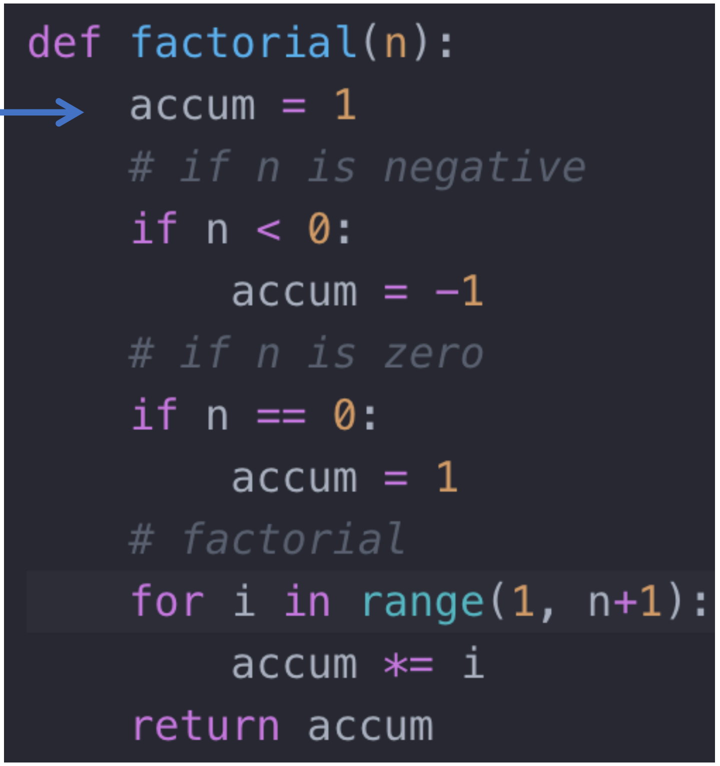
This code is good ☺ but...

No docString

Lab Recap

Let’s look at the function in a bit more depth…

Accumulator is set to 1, that’s fine

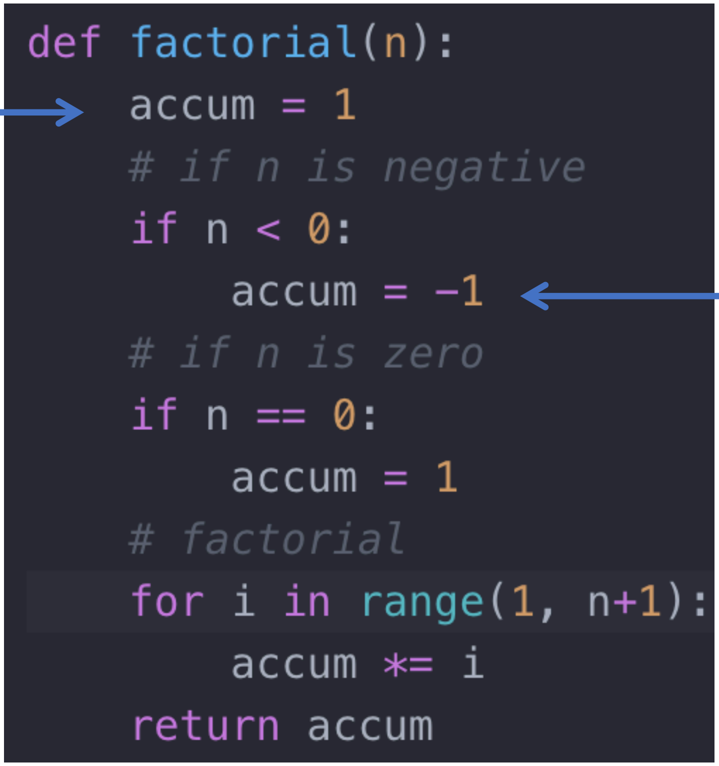


```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

Lab Recap

Let's look at the function in a bit more depth…

Accumulator is set to 1, that's fine



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

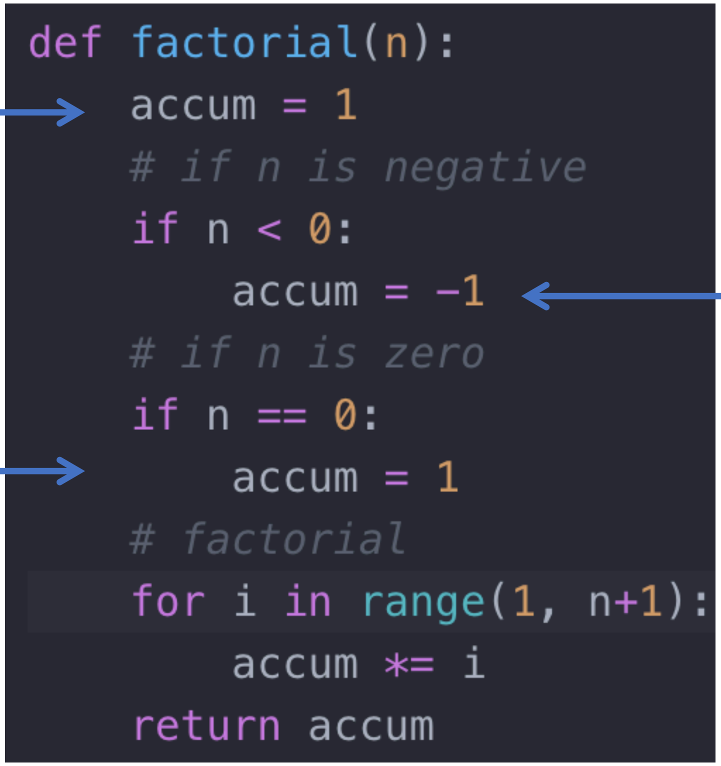
We check for negative numbers

Lab Recap

Let’s look at the function in a bit more depth…

Accumulator is set to 1, that’s fine

We check for n equal to zero



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

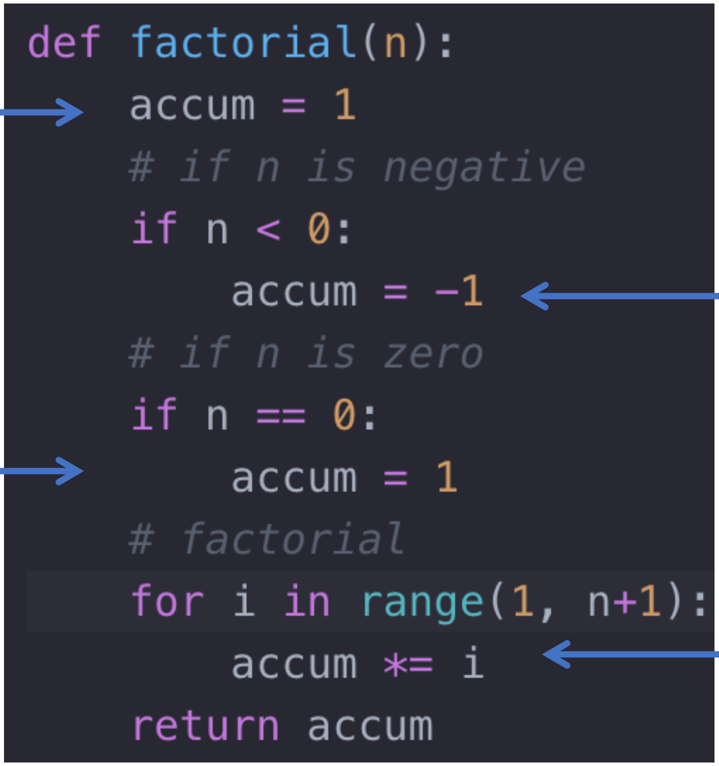
We check for negative numbers

Lab Recap

Let's look at the function in a bit more depth…

Accumulator is set to 1, that's fine

We check for n equal to zero



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

We check for negative numbers

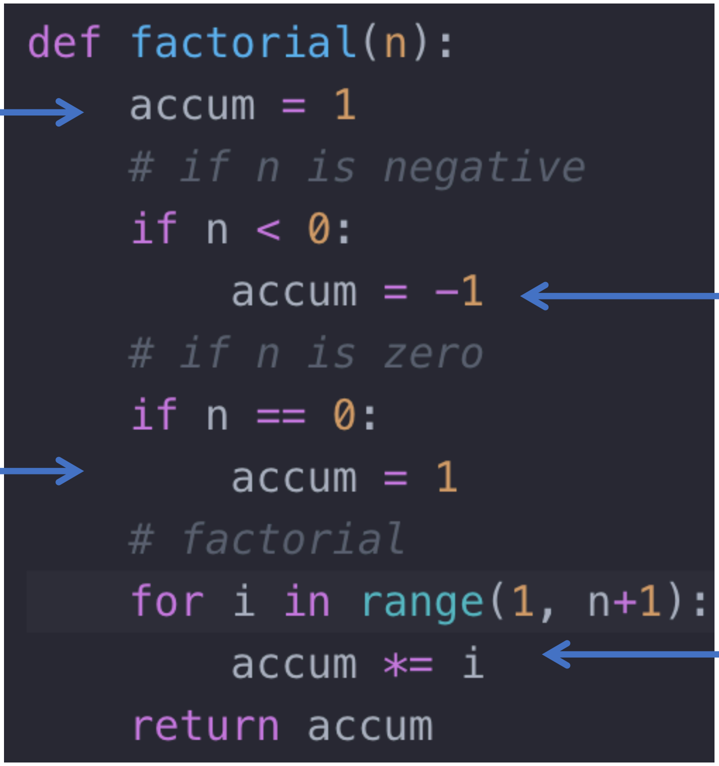
And we get our factorial for larger numbers, by using range and *=

Lab Recap

Let’s look at the function in a bit more depth…

Accumulator is set to 1, that’s fine

We check for n equal to zero



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

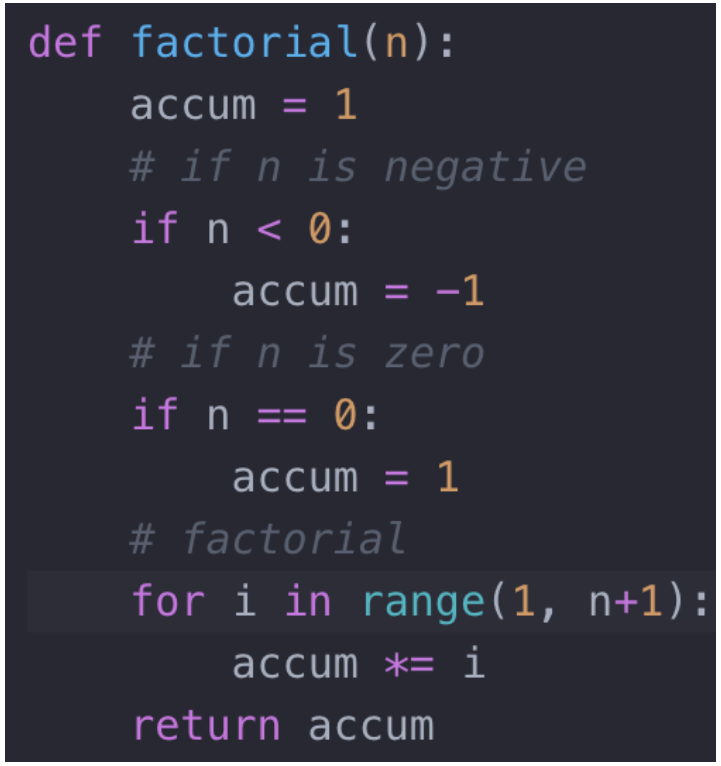
We check for negative numbers

And we get our factorial for larger numbers, by using range and *=

All requested requirements are checked for…

Lab Recap

But the code can be refined, so let’s start with returns

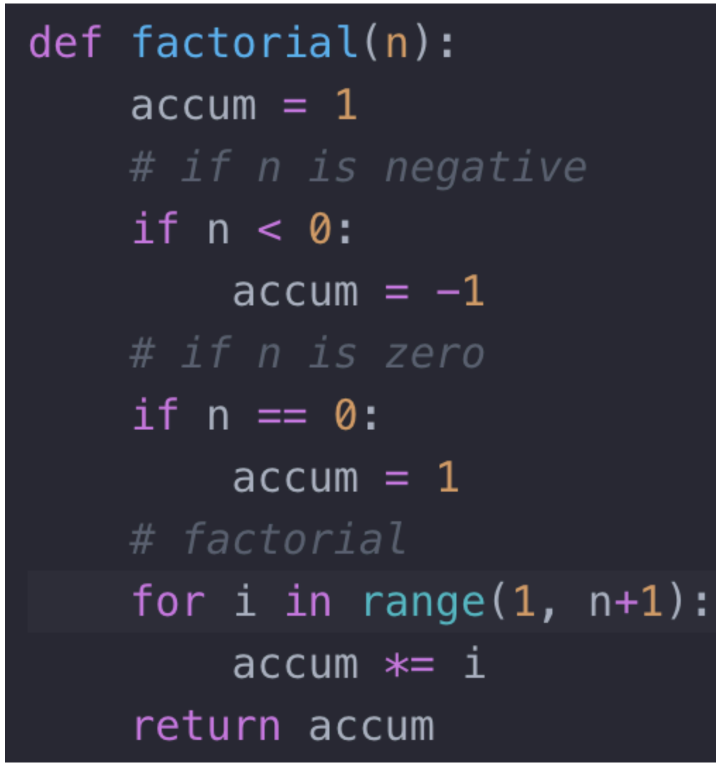


```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

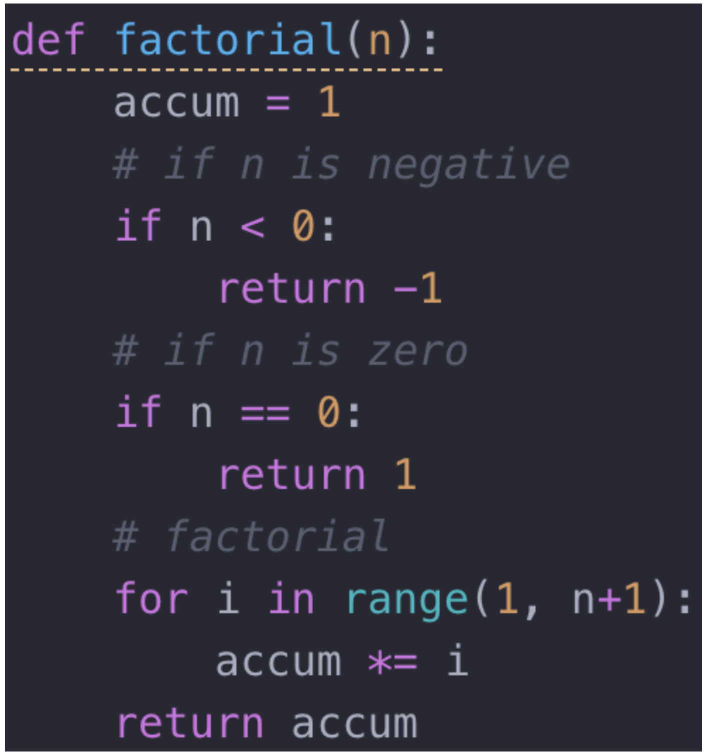
If we check for something and reset our accum, then return it

Lab Recap

But the code can be refined, so let’s start with returns



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

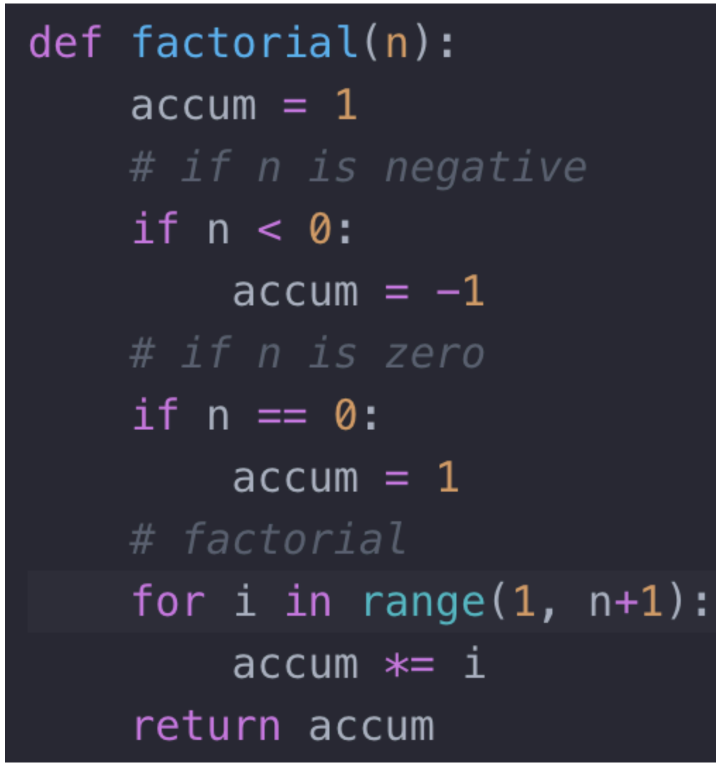


```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        return -1
    # if n is zero
    if n == 0:
        return 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

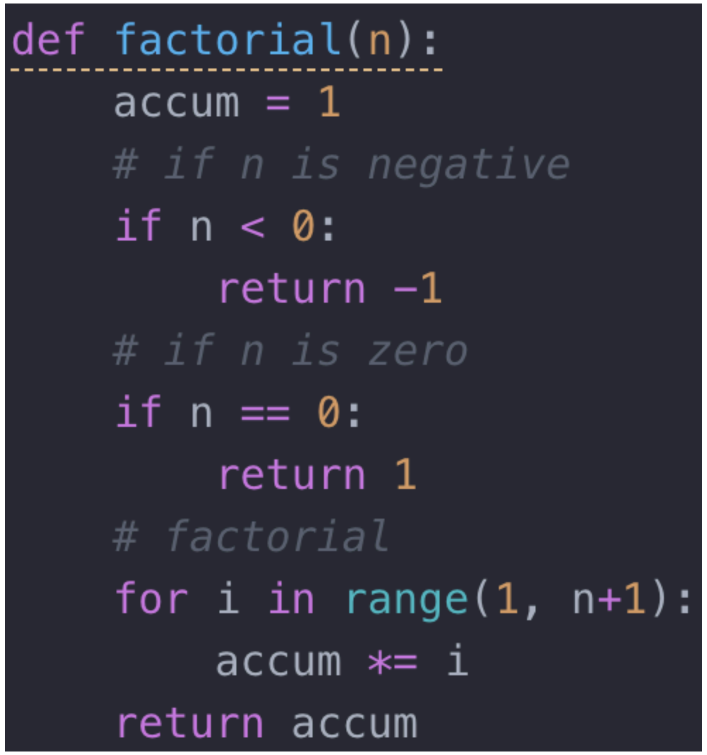
If we check for something and reset our accum, then return it

Lab Recap

But the code can be refined, so let's start with returns



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        accum = -1
    # if n is zero
    if n == 0:
        accum = 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

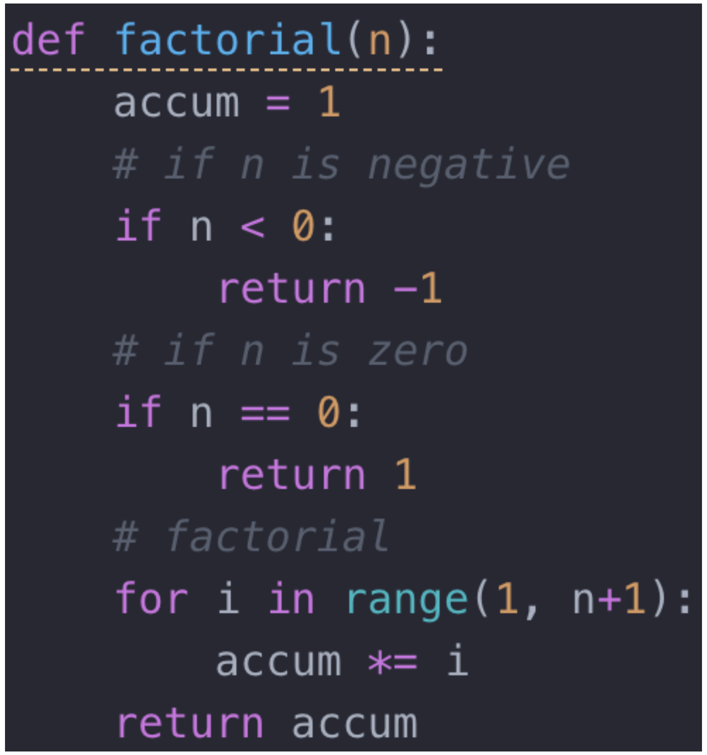


```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        return -1
    # if n is zero
    if n == 0:
        return 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

The biggest benefit from return, is that you don't have to check the later code in the statement block and it speeds up your code

Lab Recap

But the code can be further refined - if we look at range:



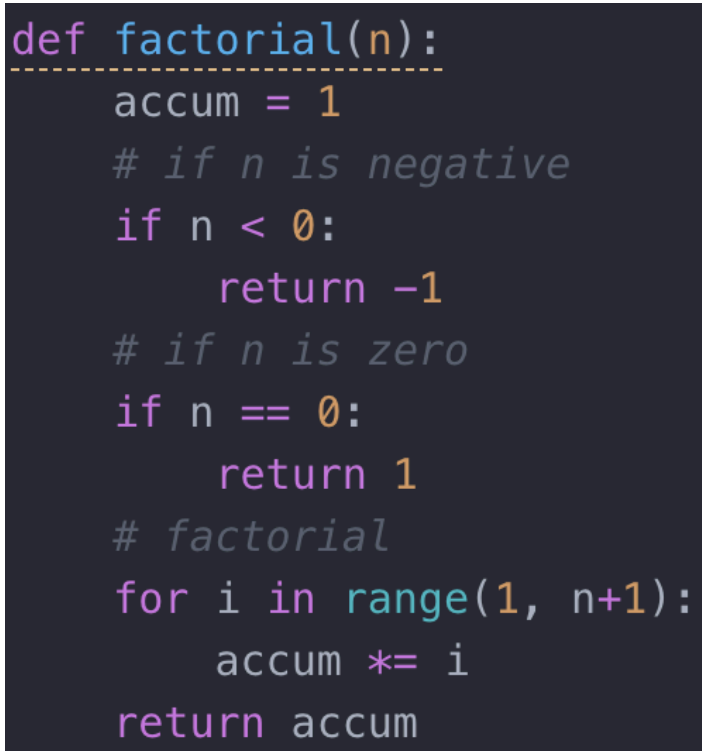
```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        return -1
    # if n is zero
    if n == 0:
        return 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

range(i, j) will return all number from i to j-1

If i >= j, nothing is returned

Lab Recap

But the code can be further refined - if we look at range:



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        return -1
    # if n is zero
    if n == 0:
        return 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

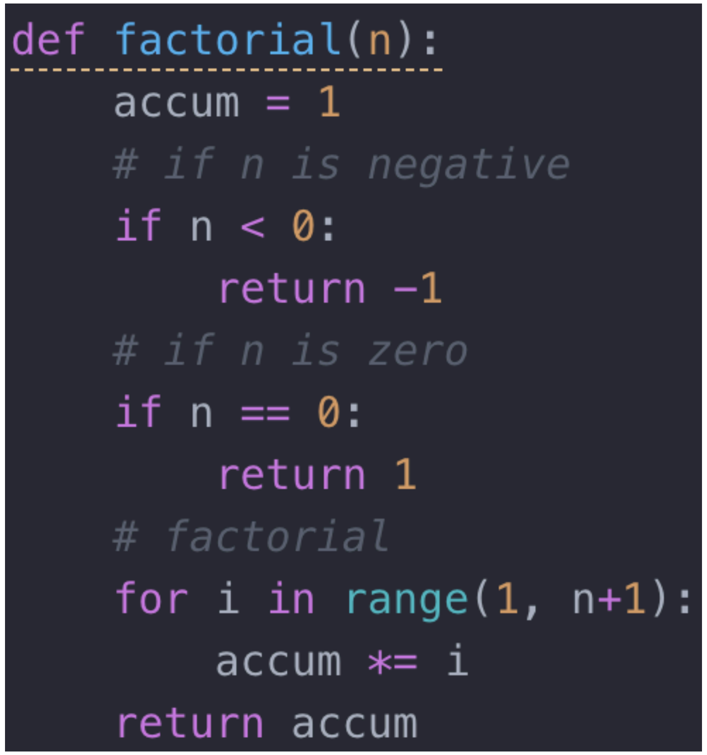
range(i, j) will return all number from i to j-1

If i >= j, nothing is returned

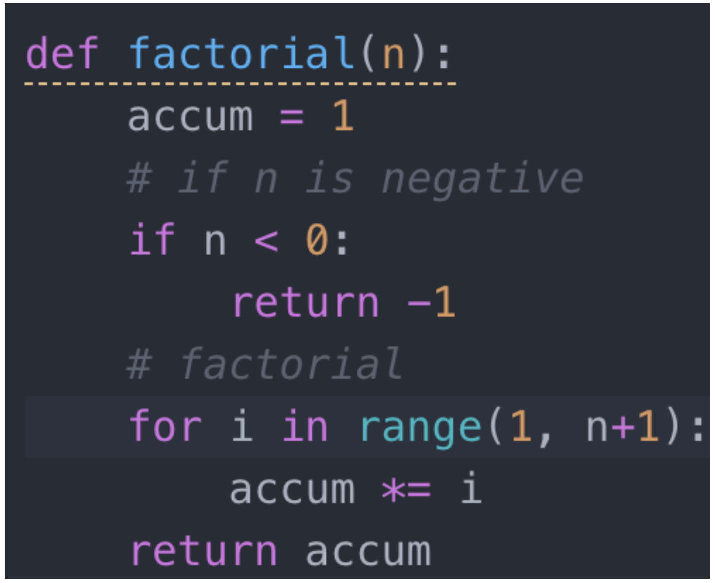
So, in our example if n == 0, then nothing is returned, and accum stays at 1, we can remove the n == 0 check

Lab Recap

But the code can be further refined - if we look at range:



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        return -1
    # if n is zero
    if n == 0:
        return 1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```



```python
def factorial(n):
    accum = 1
    # if n is negative
    if n < 0:
        return -1
    # factorial
    for i in range(1, n+1):
        accum *= i
    return accum
```

range(i, j) will return all number from i to j-1

If i >= j, nothing is returned

So, in our example if n == 0, then nothing is returned, and accum stays at 1, we can remove the n == 0 check

Lab Recap

If you have problems with a lab, ask questions…

Either in the lab or on the anonymous google form

Attend the coding class on Monday mornings, ask questions

Watch the video solution I post every Saturday

Then ask more questions the next week…

If we are in Lab 8 and you have a question on Lab 3, then ask

Lab Recap

Finally:

Do not copy other peoples code

I am still seeing the same errors popping up…

I am still seeing the same coding style, same comments, near exact same code…

UCC

University College Cork, Ireland

Coláiste na hOllscoile Corcaigh