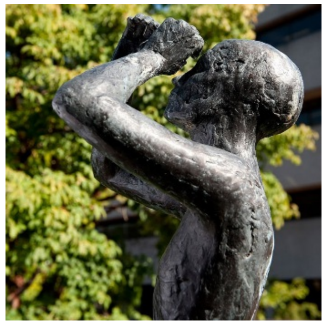


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Announcements

Note: the PCs in G20

As stated, you will need your CS account details

to log in to the PCs in G20

David O’Byrne, IT Manager, will come to G20 at 4pm tomorrow and Wednesday, and will give a little talk on the CS IT helpdesk.

He will also hand out any remaining accounts which have not as yet been registered.

Go over common issues, etc.

Announcements

Note: the PCs in G20

You have an allocated virtual hard drive of say 5GB. If you go over this allocation, normally the first issue is you can’t log in

So:

1. Don’t try to install something that is already installed (say installing PyCharm when it is already there)
2. Make sure you know what’s installed on the machines and what you need to use these apps (code, packages, etc.)
3. Wait until your first lab, when your lecturer will explain what to do, how to do and when to do it.
4. If like me, they mention software we are going to use, then install this on your own home machine/laptop, but until told to by the lecturer do not try and install the software on the G20 machines.

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

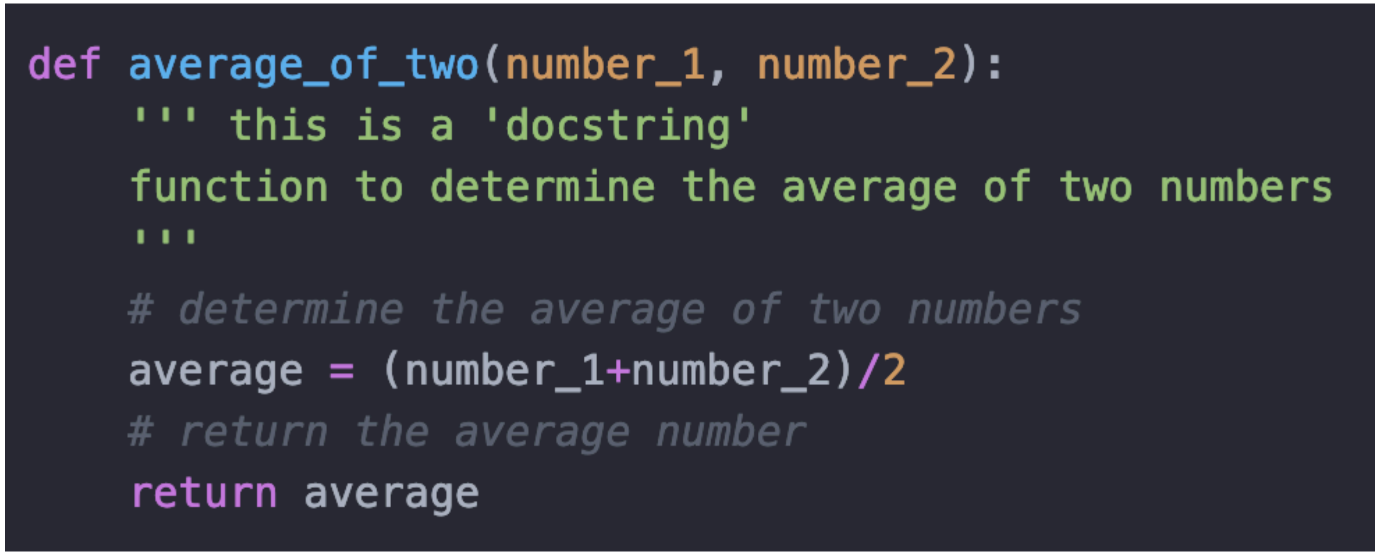
Python Functions Recap

- We wrote an “average_of_two” function
  - That calculates the average of two numbers
  - And returns said average
- We add a docstring comment, which is viewable in an IDE
- We noted that our returned value was a float
  - Generating a value with a decimal place
- We looked at Python Operators
  - (=), (+), (/), (*), (-), (**), (%) and (//)
  - We noted that the operators are mutable
    - Change their operation based on the data type they are working with
  - They have precedence (similar to BOMDAS)

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Functions

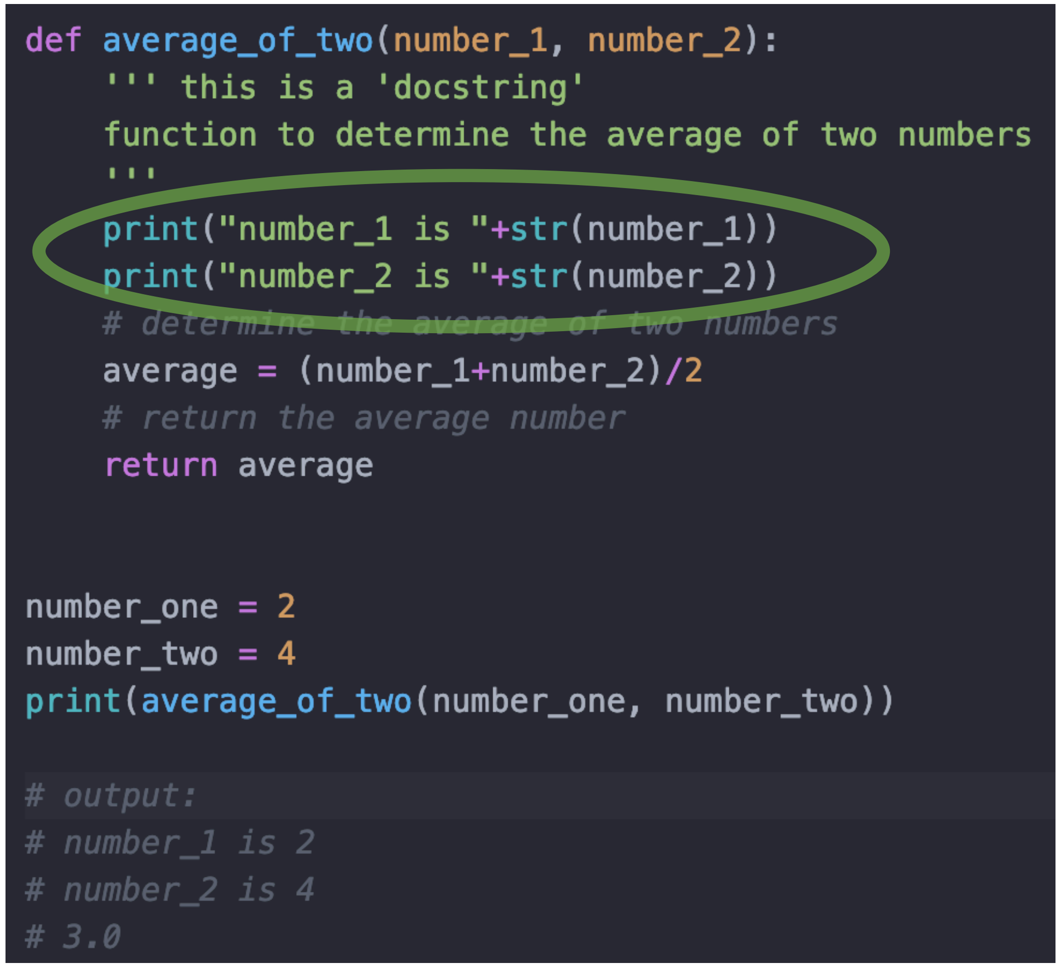
Let’s look at out new average_of_two function one last time



```python
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average
```

Python Functions

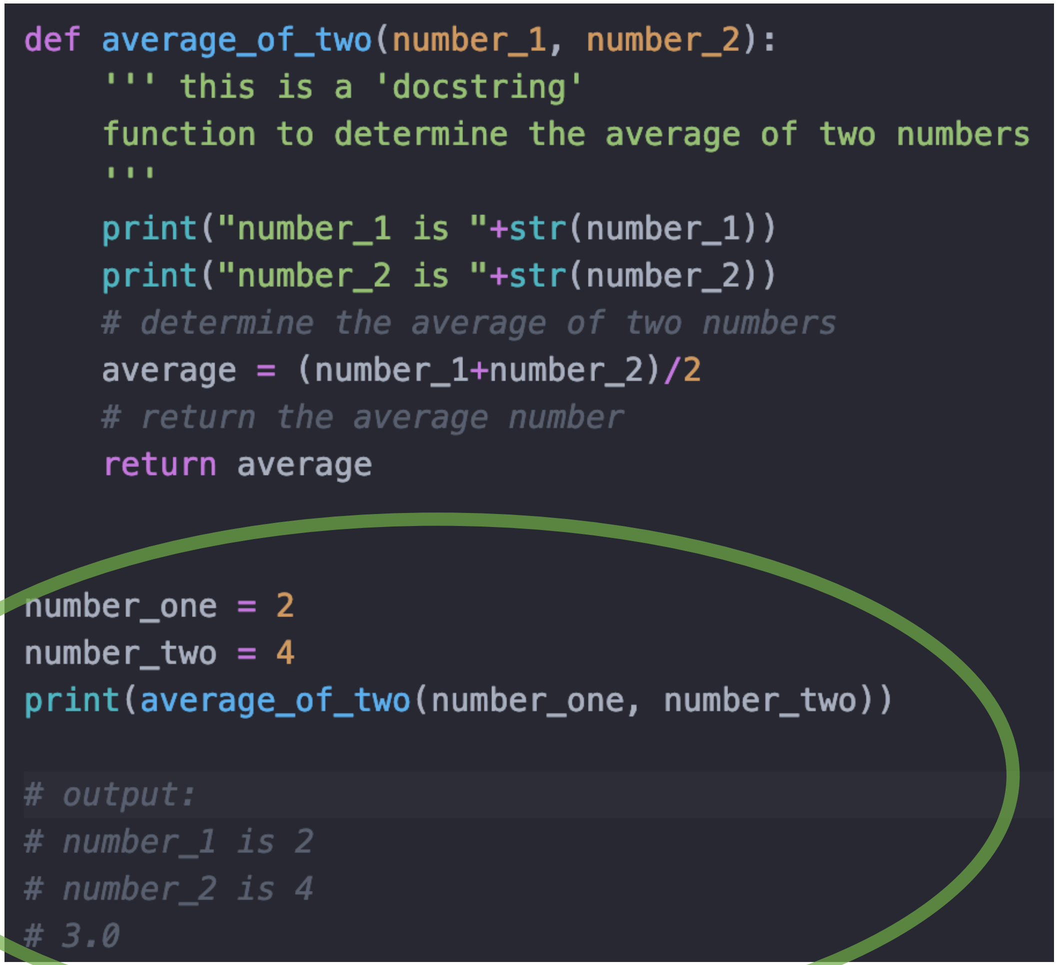
Let’s add some printouts to see the variable values being passed in



```python
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    print("number_1 is "+str(number_1))
    print("number_2 is "+str(number_2))
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


number_one = 2
number_two = 4
print(average_of_two(number_one, number_two))

# output:
# number_1 is 2
# number_2 is 4
# 3.0
```

Python Functions

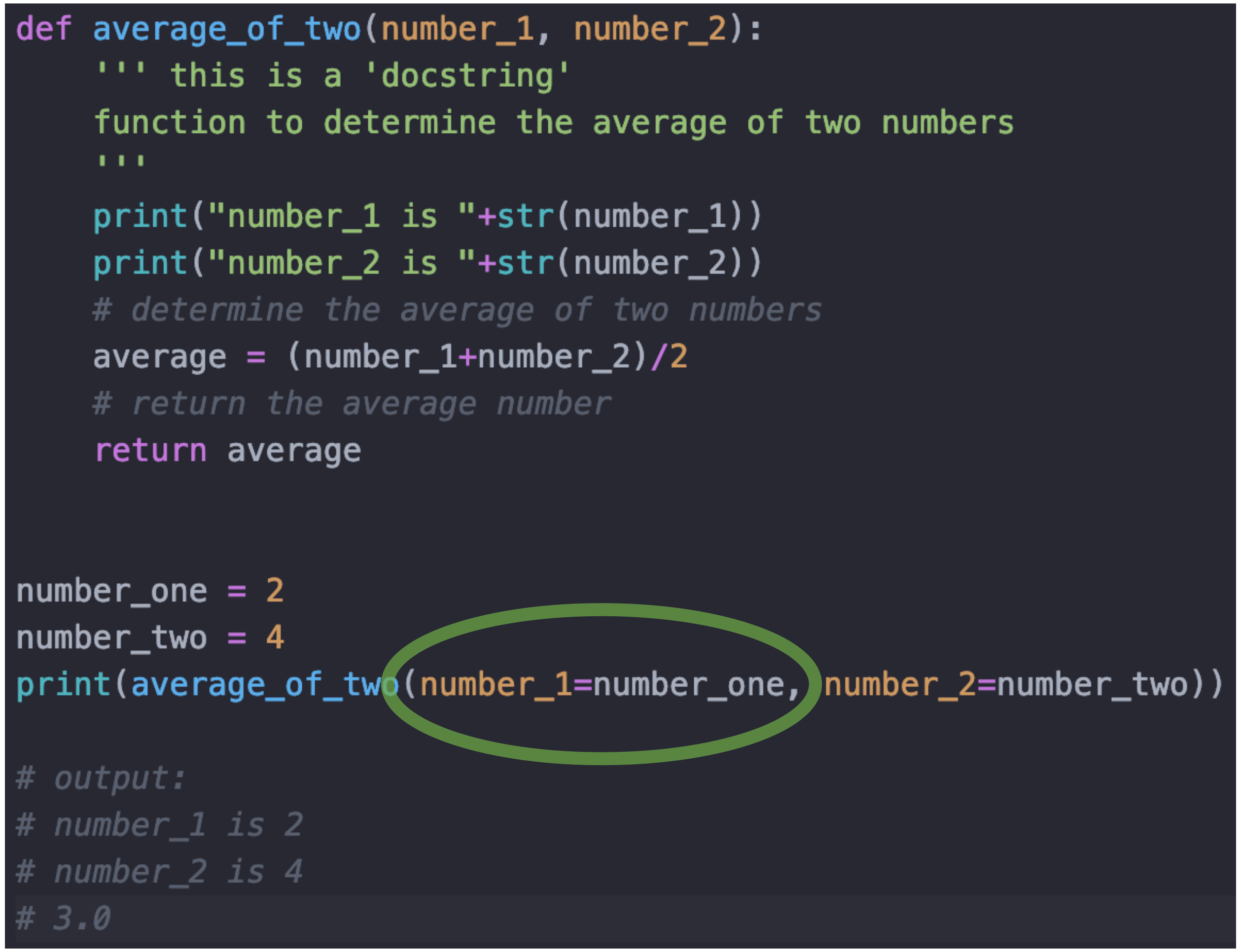
Let's call the function and view the results



```python
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    print("number_1 is "+str(number_1))
    print("number_2 is "+str(number_2))
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


number_one = 2
number_two = 4
print(average_of_two(number_one, number_two))

# output:
# number_1 is 2
# number_2 is 4
# 3.0
```

Python Functions

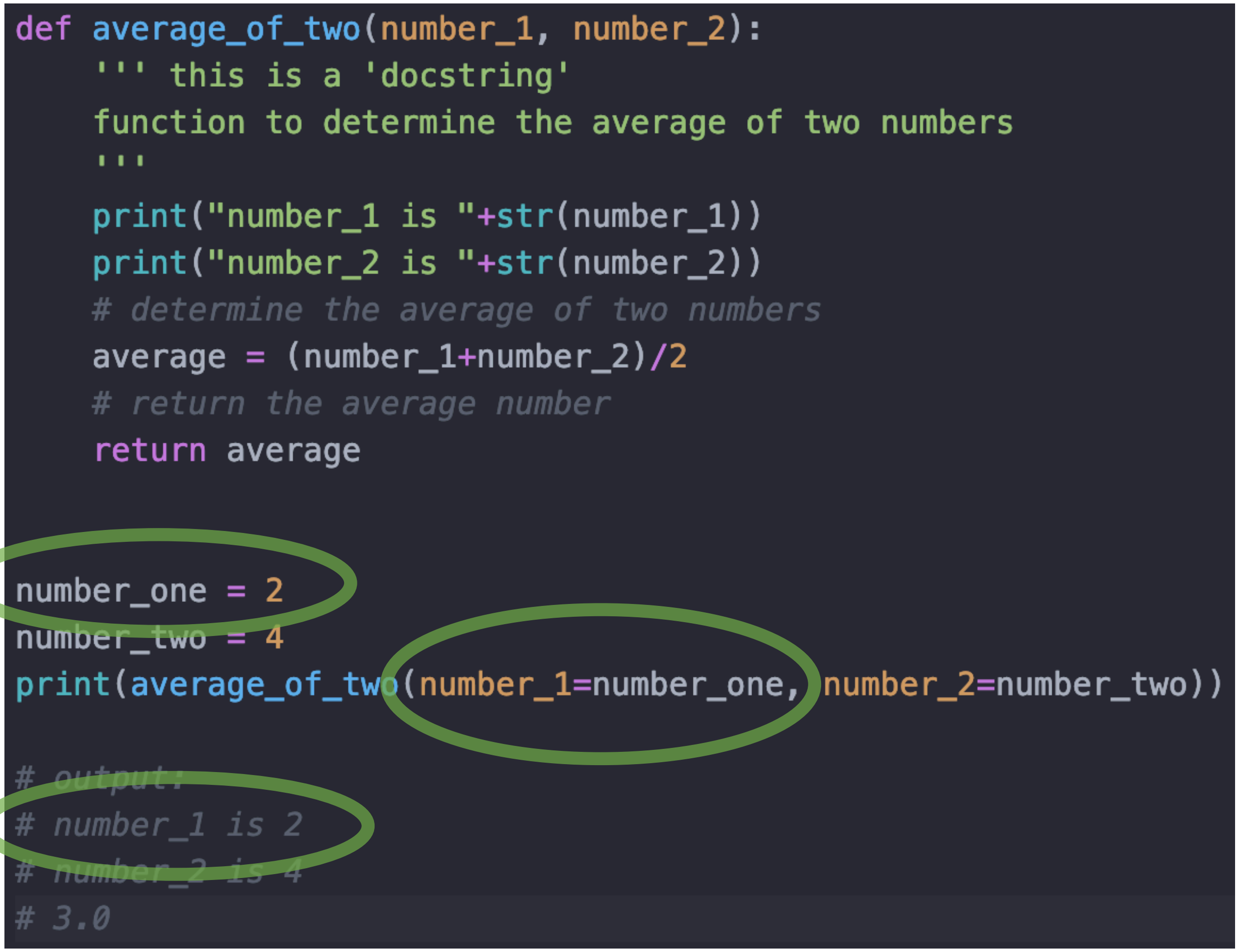
We can also choose which parameter gets which input value



```python
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    print("number_1 is "+str(number_1))
    print("number_2 is "+str(number_2))
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


number_one = 2
number_two = 4
print(average_of_two(number_1=number_one, number_2=number_two))

# output:
# number_1 is 2
# number_2 is 4
# 3.0
```

Python Functions

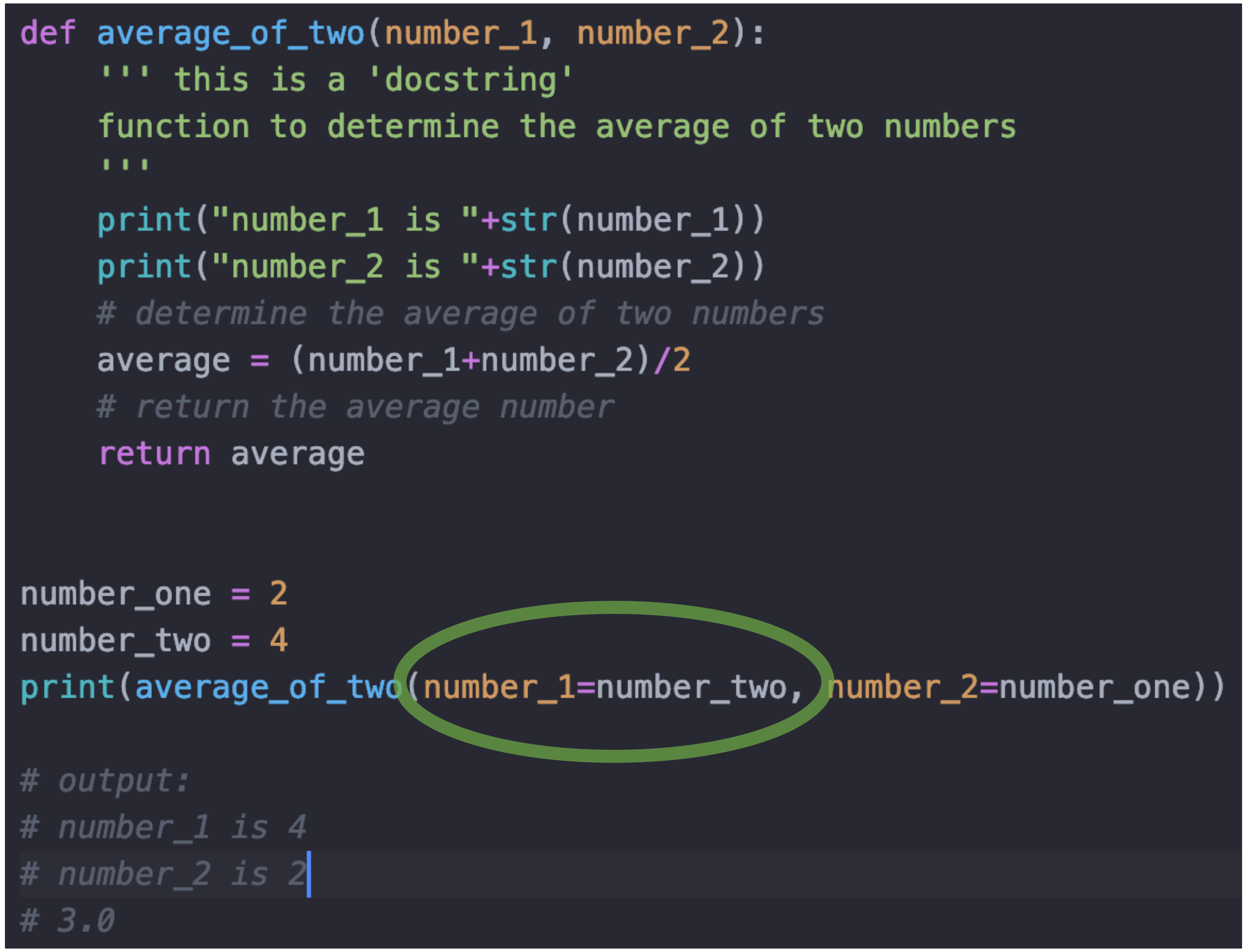
We can also choose which parameter gets which input value



```
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    print("number_1 is "+str(number_1))
    print("number_2 is "+str(number_2))
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


number_one = 2
number_two = 4
print(average_of_two(number_1=number_one, number_2=number_two))

# output:
# number_1 is 2
# number_2 is 4
# 3.0
```

Python Functions

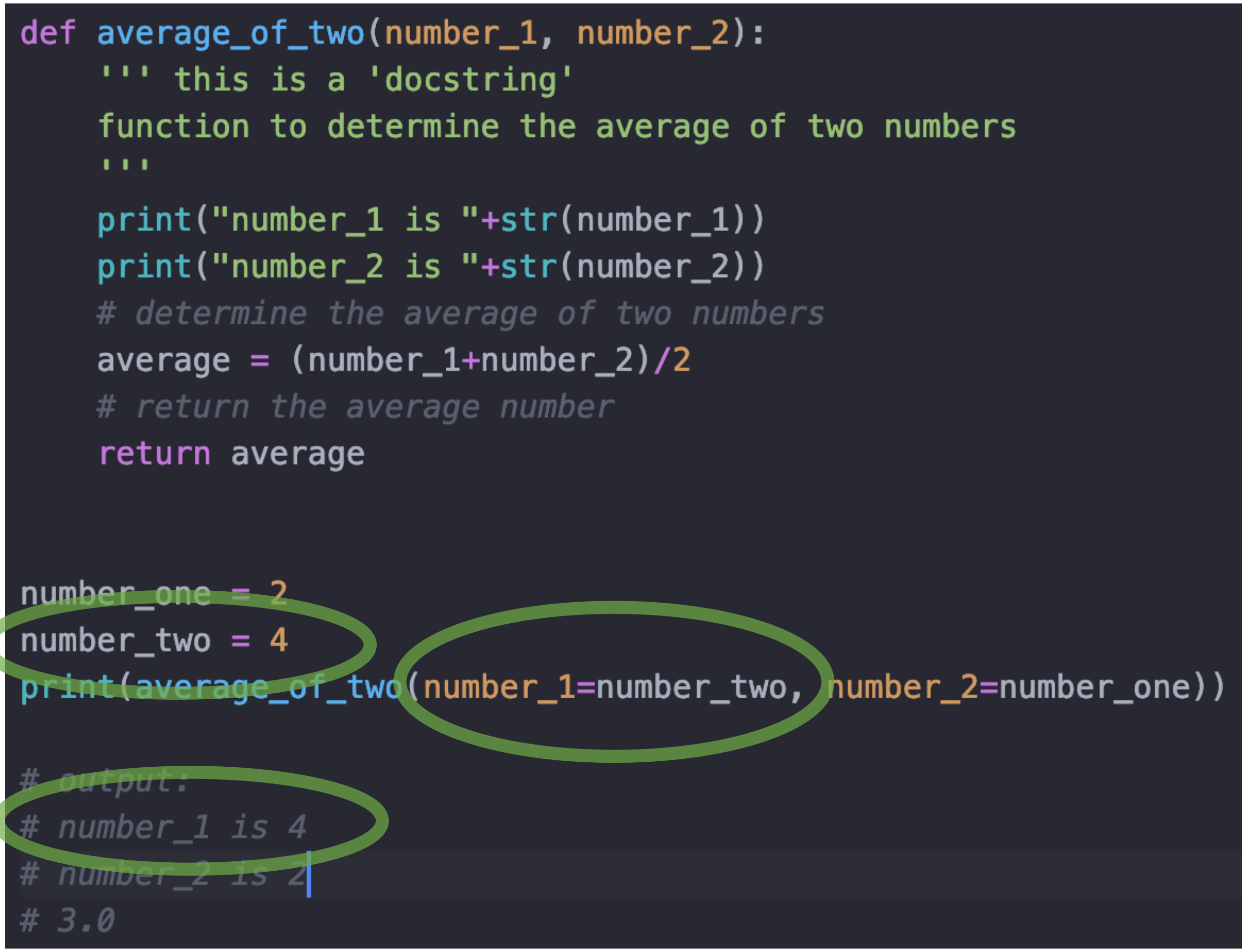
Here we swop the input variables and the parameter values reverse



```
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    print("number_1 is "+str(number_1))
    print("number_2 is "+str(number_2))
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


number_one = 2
number_two = 4
print(average_of_two(number_1=number_two, number_2=number_one))

# output:
# number_1 is 4
# number_2 is 2
# 3.0
```

Python Functions

Here we swop the input variables and the parameter values reverse

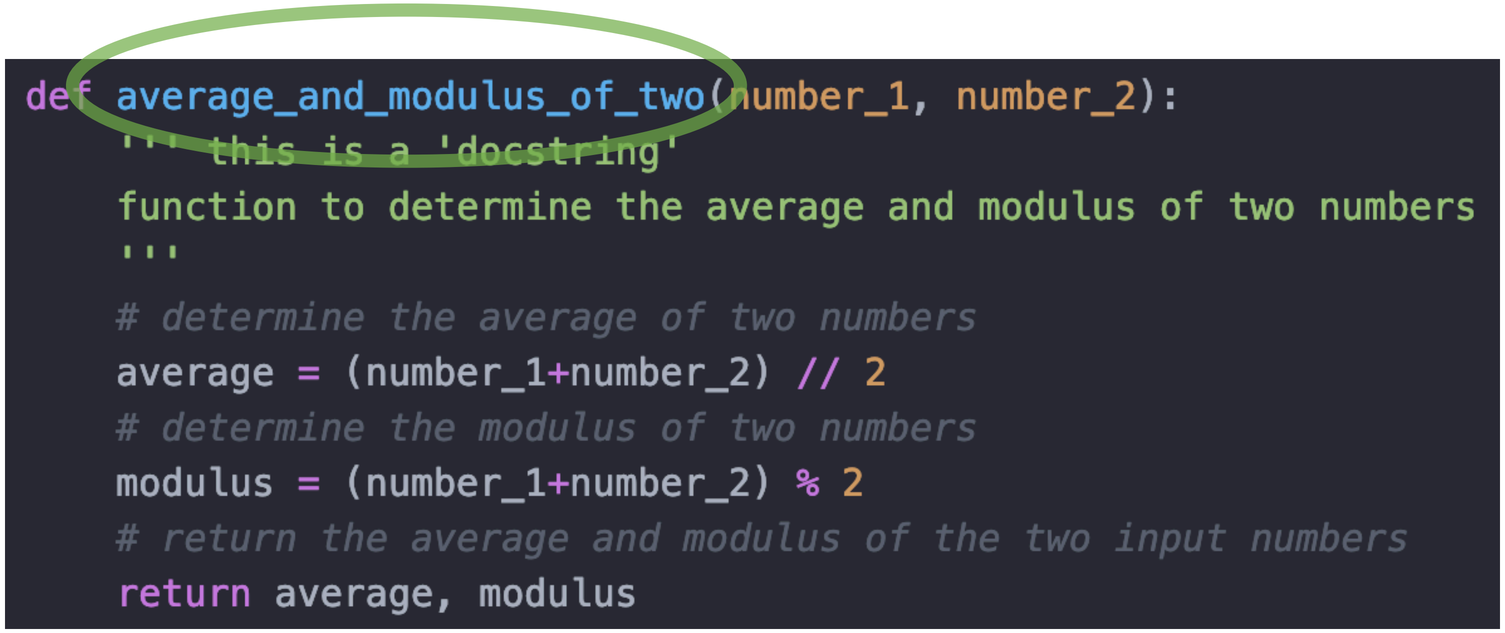


```
def average_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average of two numbers
    '''
    print("number_1 is "+str(number_1))
    print("number_2 is "+str(number_2))
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


number_one = 2
number_two = 4
print(average_of_two(number_1=number_two, number_2=number_one))

# output:
# number_1 is 4
# number_2 is 2
# 3.0
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Functions

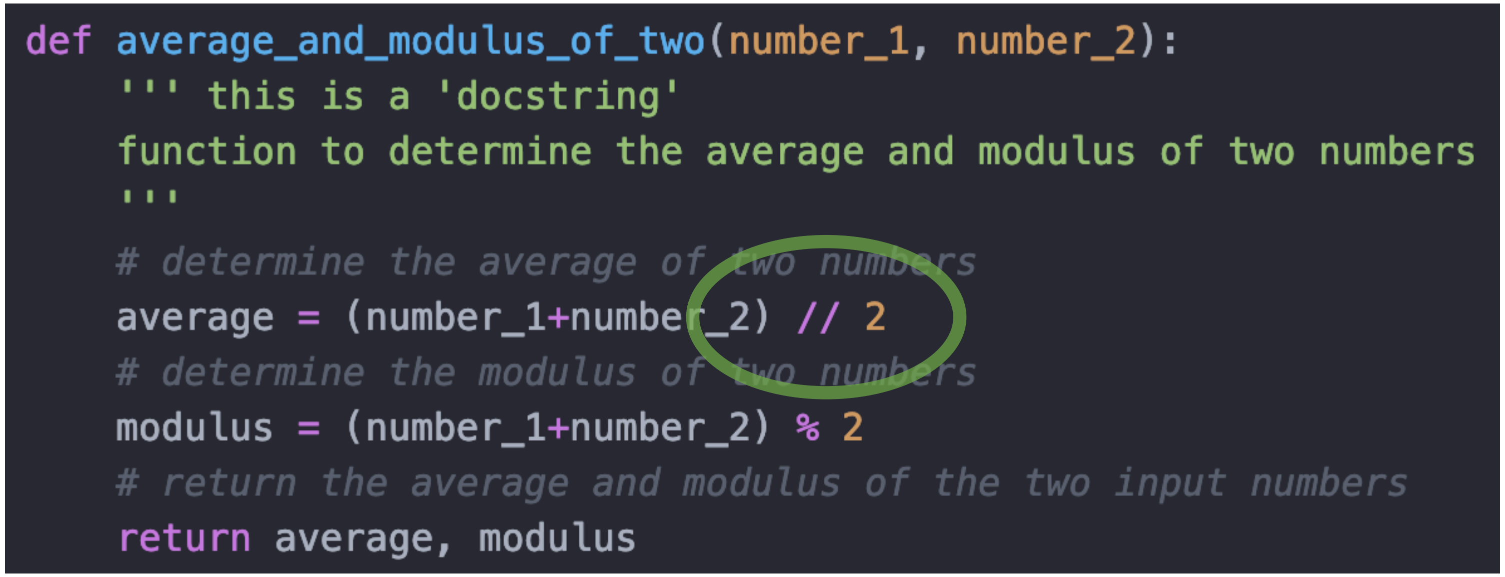
Now we have the average, let’s get the modulus (remainder)



```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus
```

Python Functions

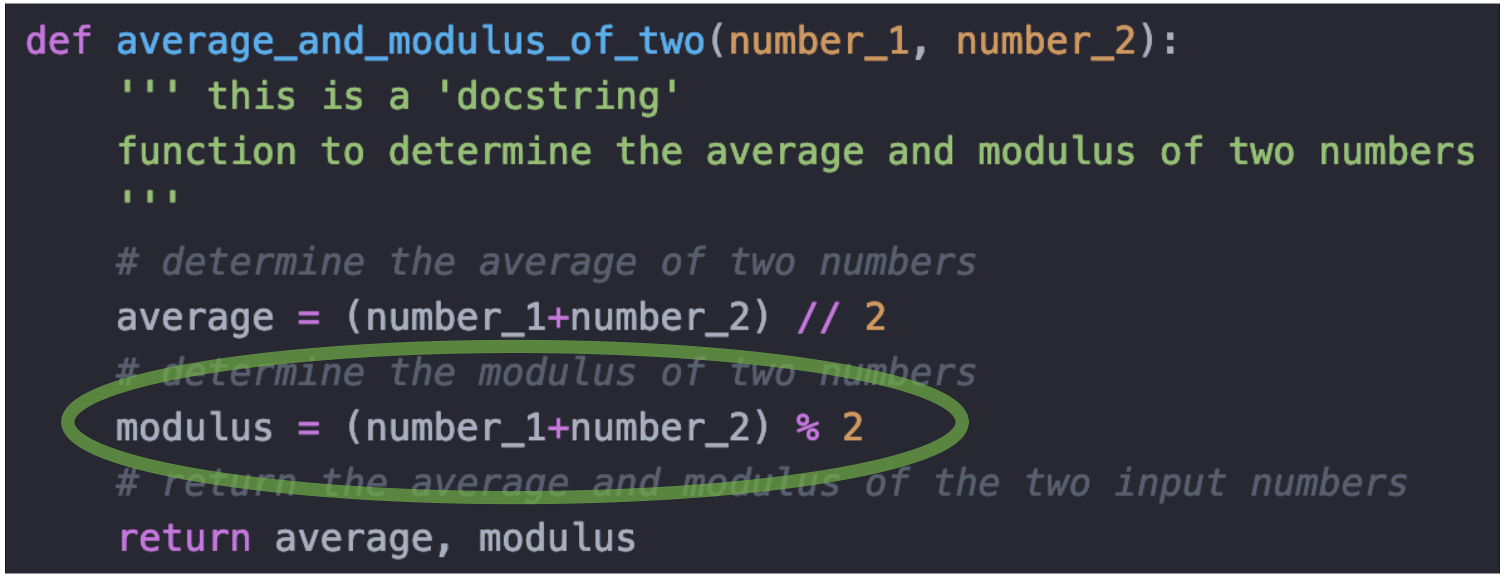
change the float division to integer division, so no more decimal places



```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus
```

Python Functions

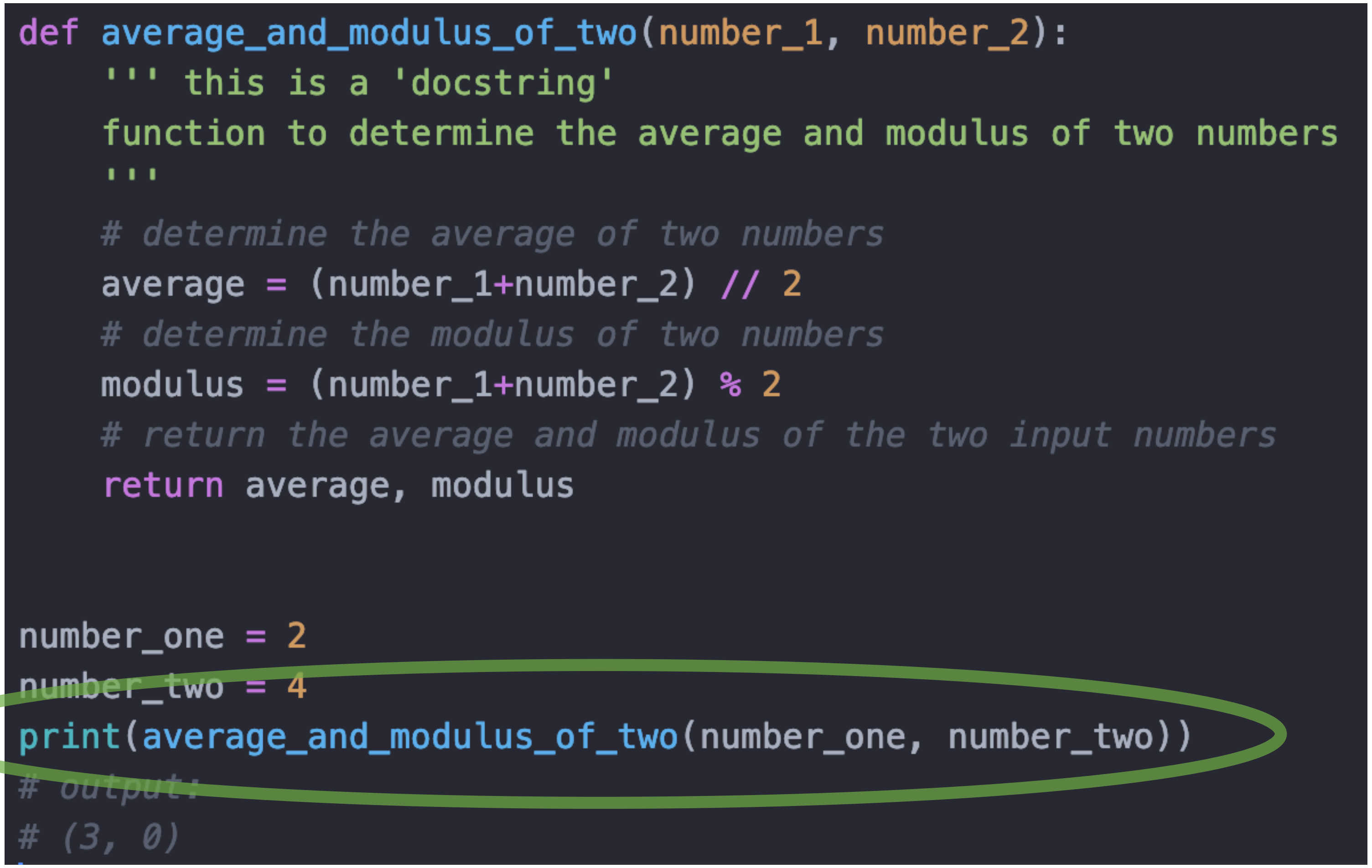
Add code to get the modulus (remainder)



```
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus
```

Python Functions

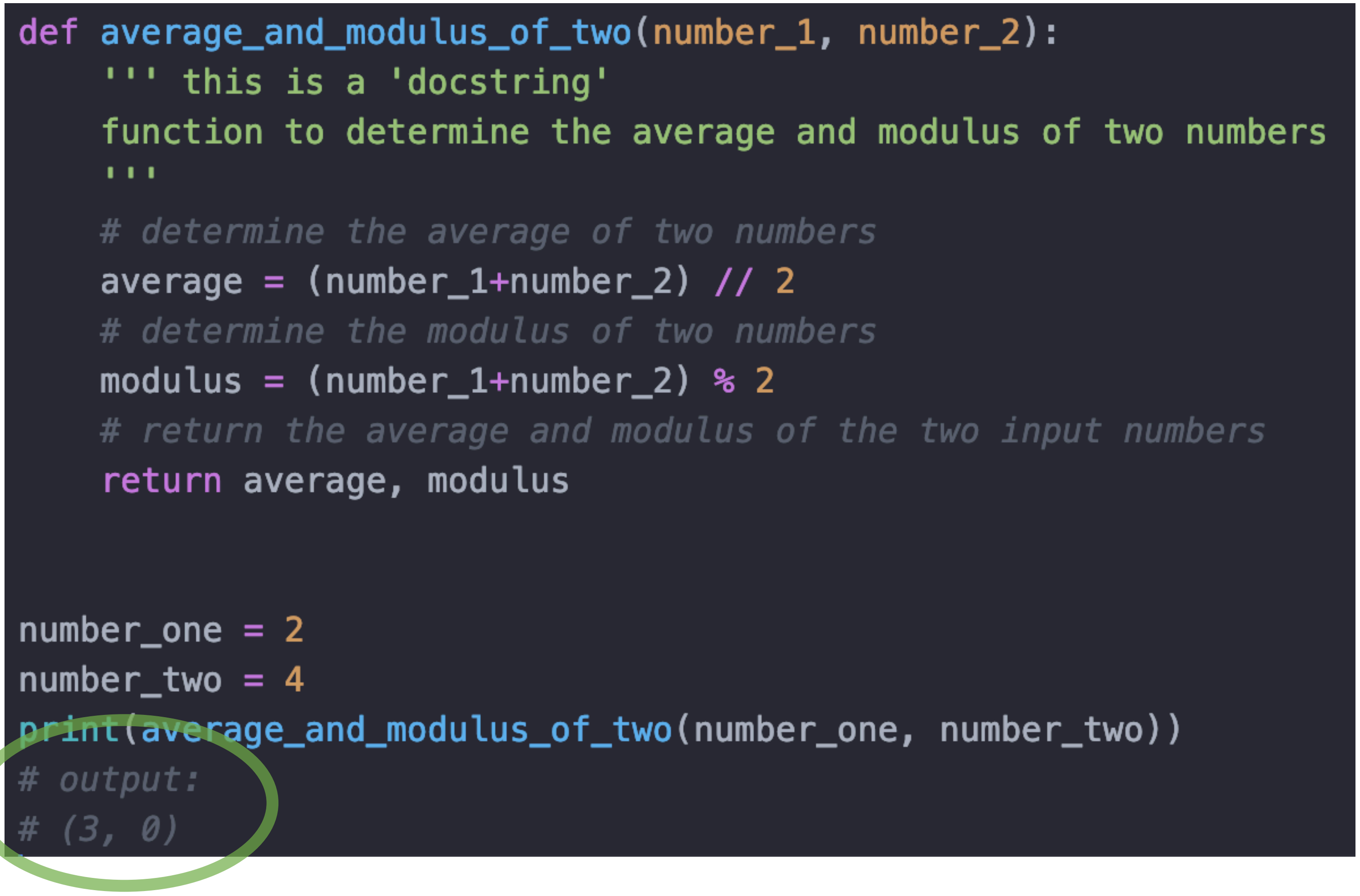
Call the new function



```
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)
```

Python Functions

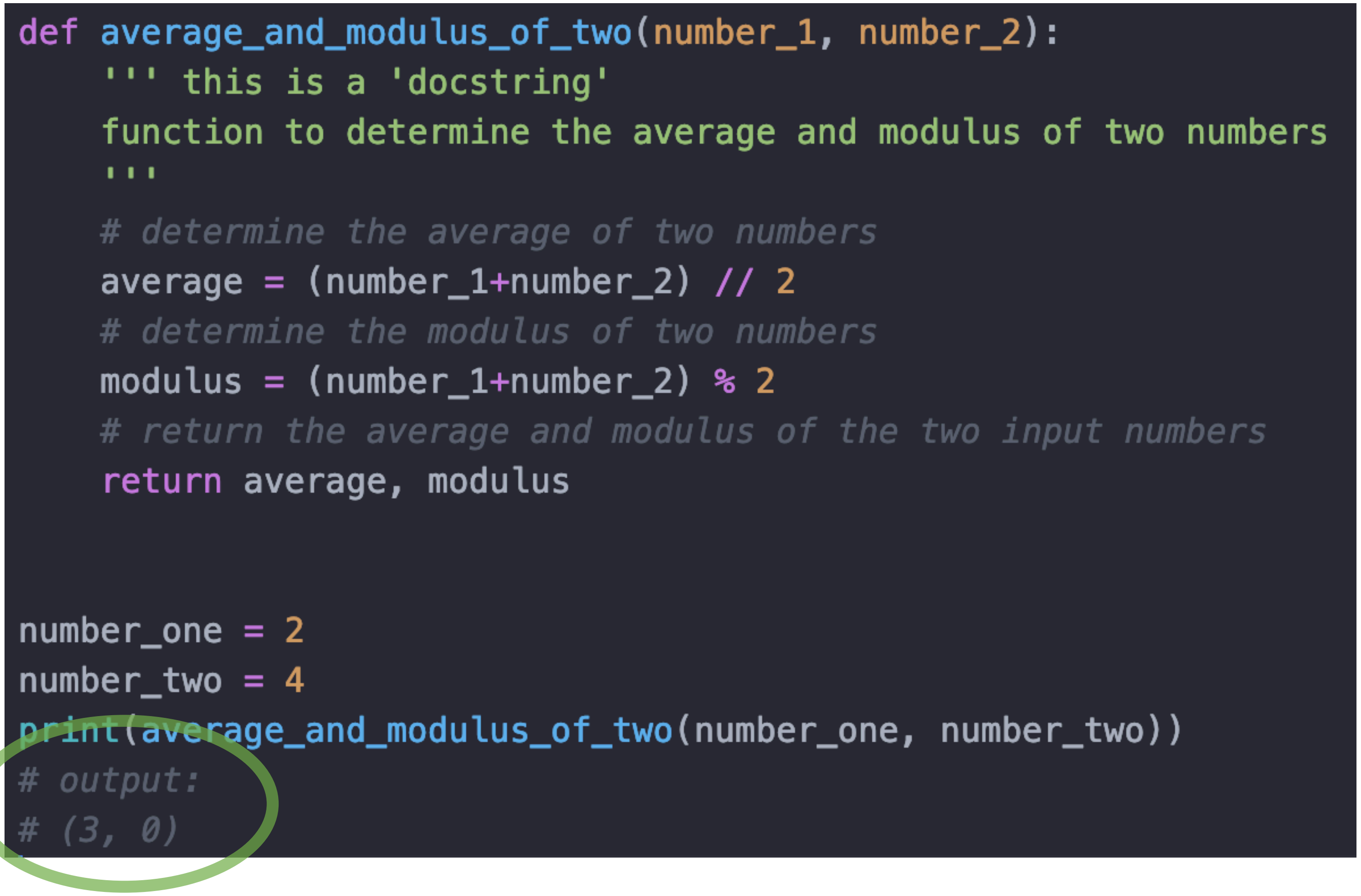
Get the output from the new function



```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)
```

Python Functions

It’s a

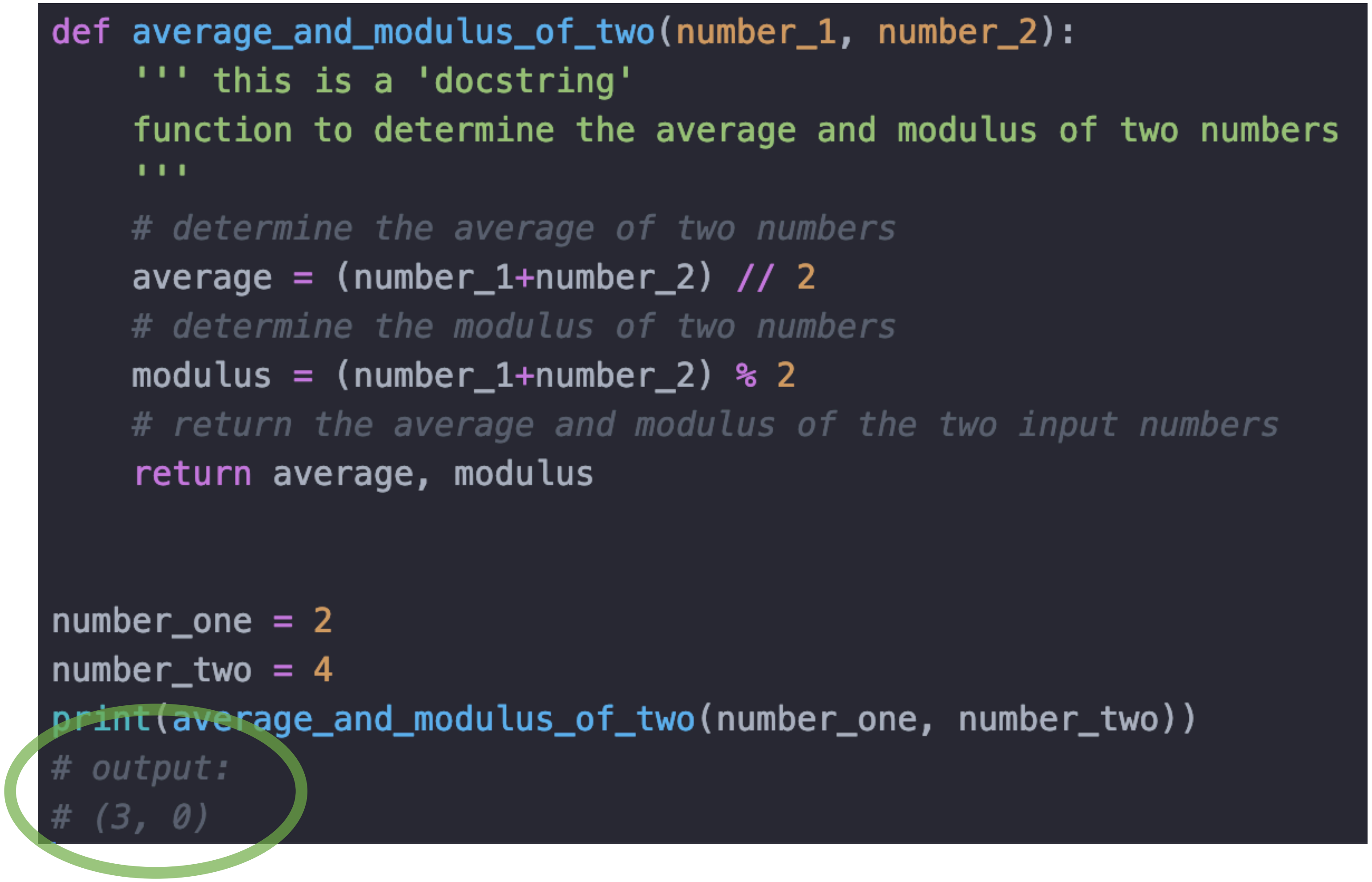


```
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Functions

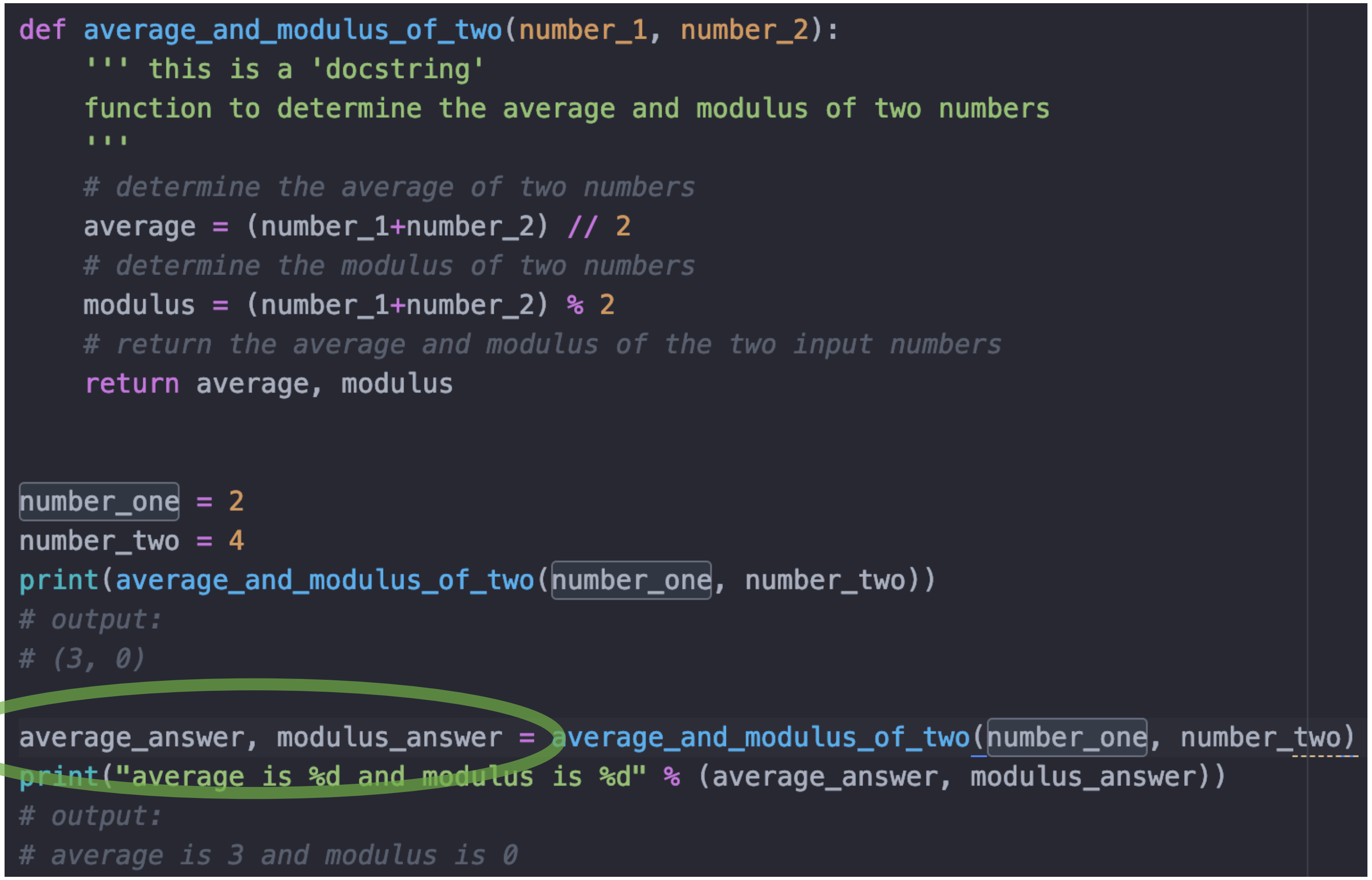
It’s a Tuple - Mind blown ☺



```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)
```

Python Functions

We can assign the values directly to variables

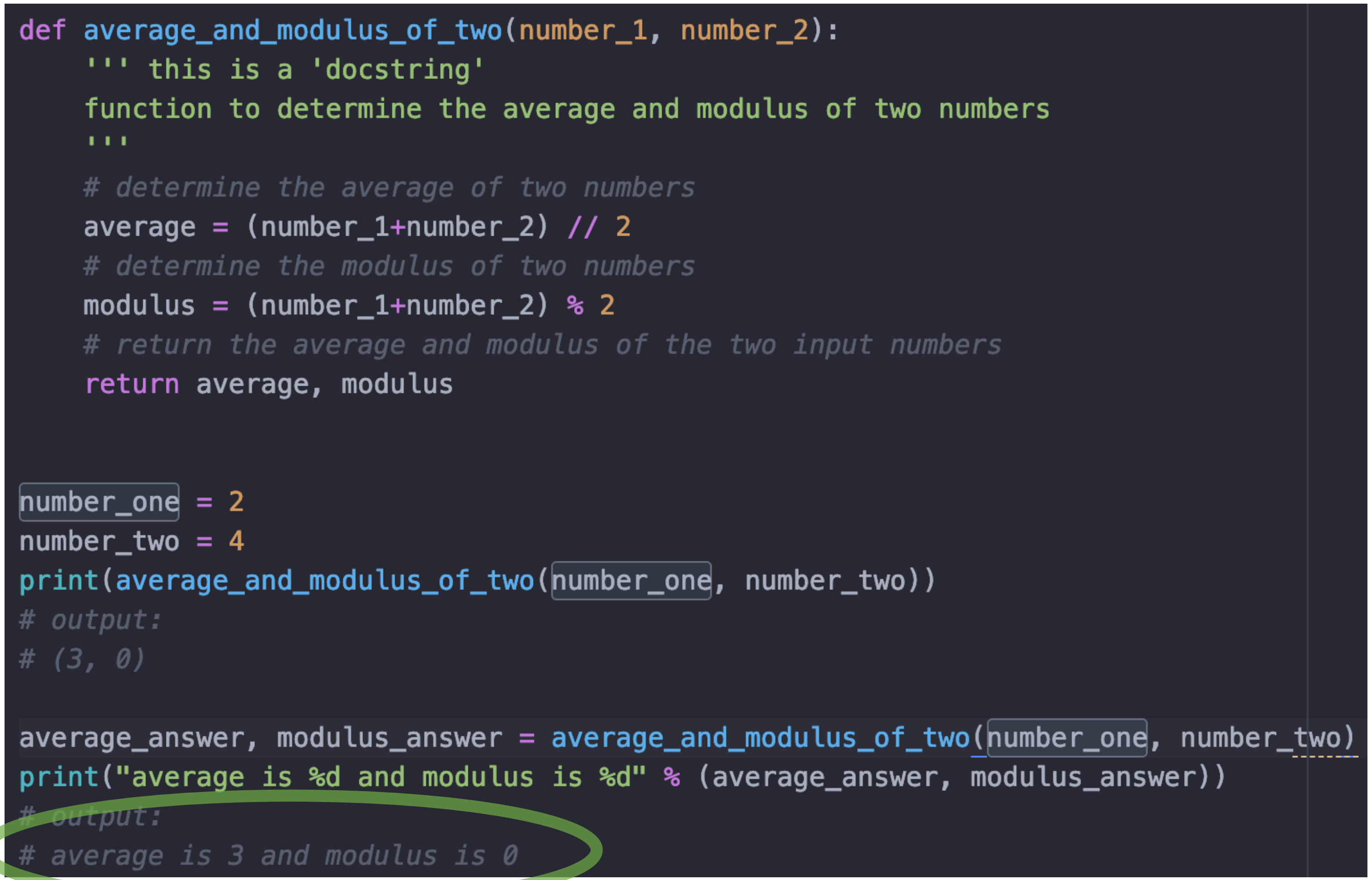


```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)

average_answer, modulus_answer = average_and_modulus_of_two(number_one, number_two)
print("average is %d and modulus is %d" % (average_answer, modulus_answer))
# output:
# average is 3 and modulus is 0
```

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Functions

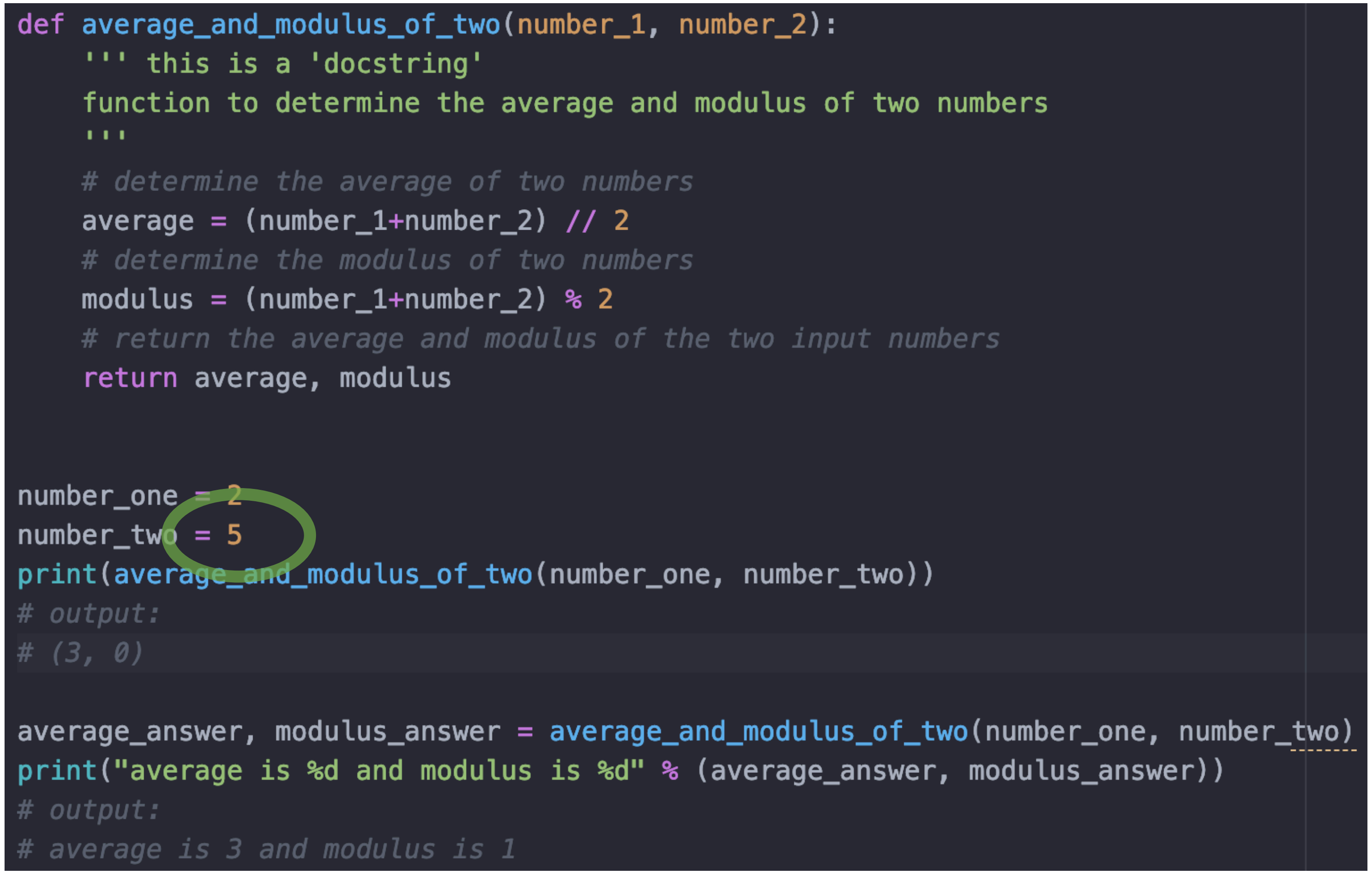
We can assign the values directly to variables



```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)

average_answer, modulus_answer = average_and_modulus_of_two(number_one, number_two)
print("average is %d and modulus is %d" % (average_answer, modulus_answer))
# output:
# average is 3 and modulus is 0
```

Python Functions

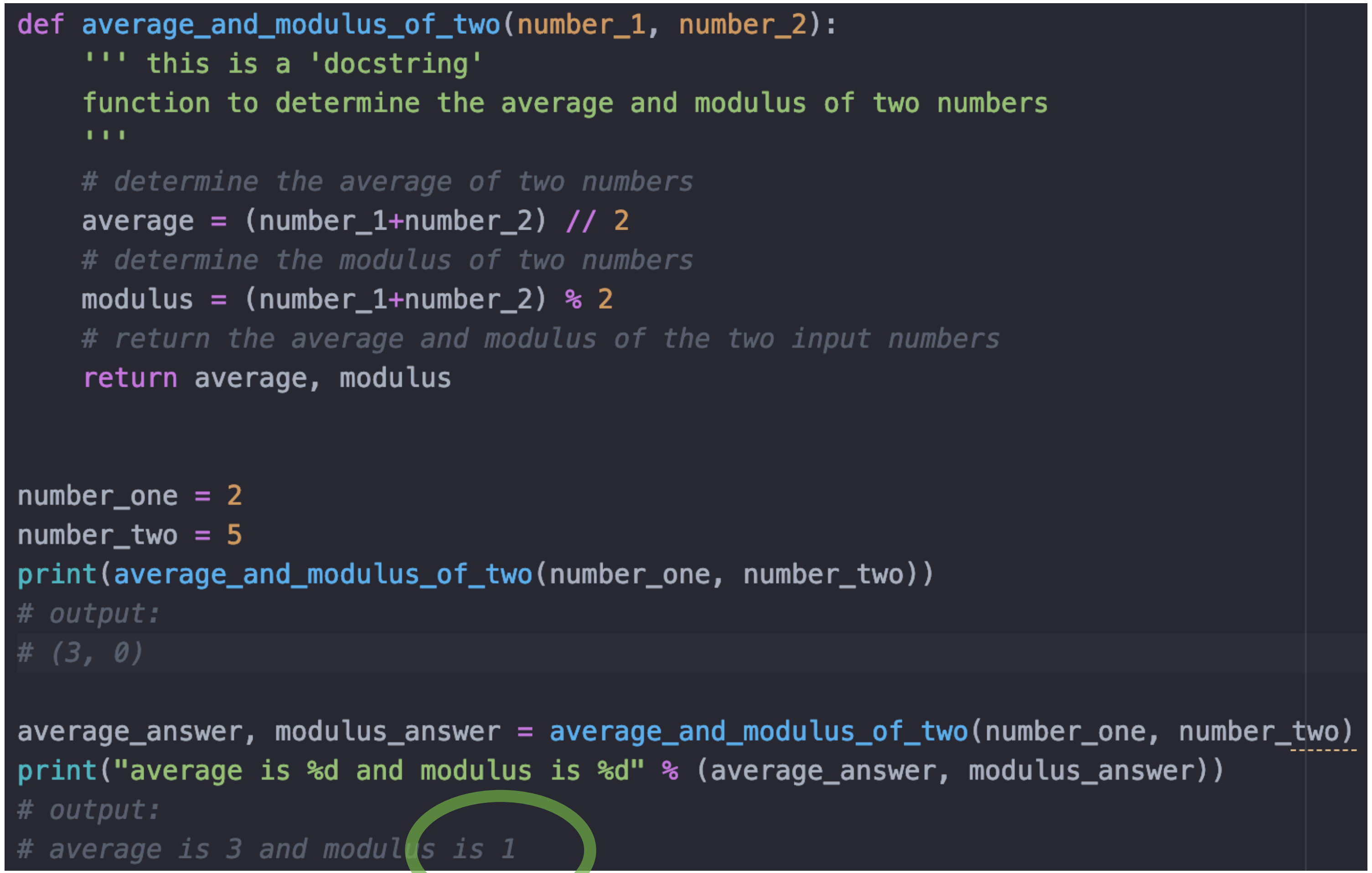
If we make one of the number odd



```python
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 5
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)

average_answer, modulus_answer = average_and_modulus_of_two(number_one, number_two)
print("average is %d and modulus is %d" % (average_answer, modulus_answer))
# output:
# average is 3 and modulus is 1
```

Python Functions

We see modulus becomes 1

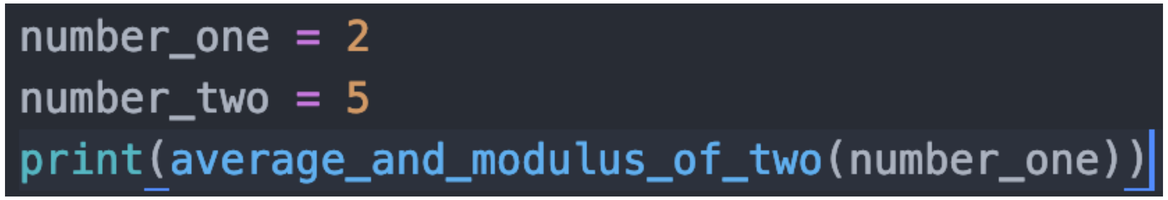


```
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 5
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)

average_answer, modulus_answer = average_and_modulus_of_two(number_one, number_two)
print("average is %d and modulus is %d" % (average_answer, modulus_answer))
# output:
# average is 3 and modulus is 1
```

Python Functions

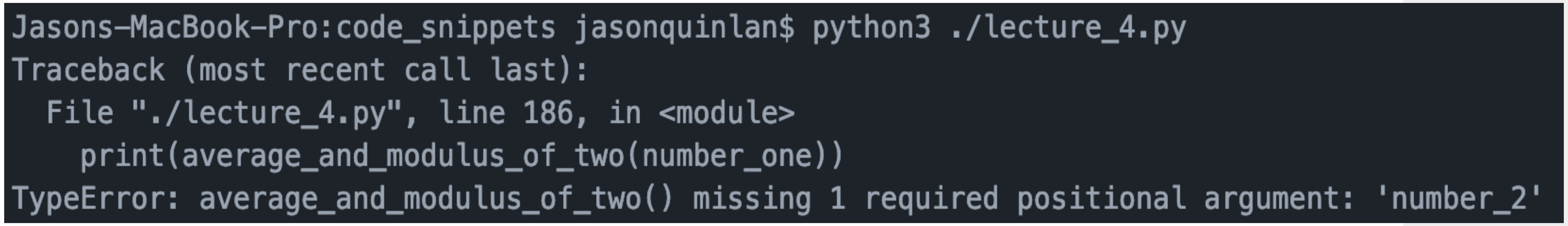
If we run the code with only one parameter to “average_and_modulus_of_two()”

If we wanted to know if a number was odd??



```
number_one = 2
number_two = 5
print(average_and_modulus_of_two(number_one))
```

We get an error

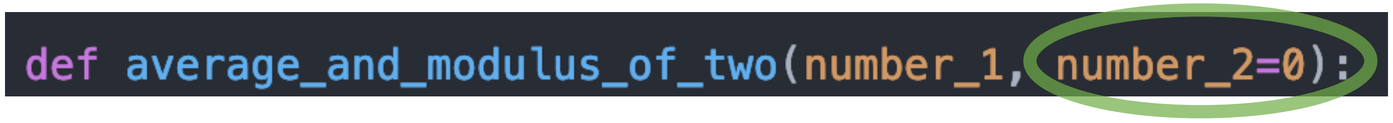


```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_4.py
Traceback (most recent call last):
  File "./lecture_4.py", line 186, in <module>
    print(average_and_modulus_of_two(number_one))
TypeError: average_and_modulus_of_two() missing 1 required positional argument: 'number_2'
```

Python Functions

To overcome the error we can set a default value to number_2

In this instance we set it to “0” - zero

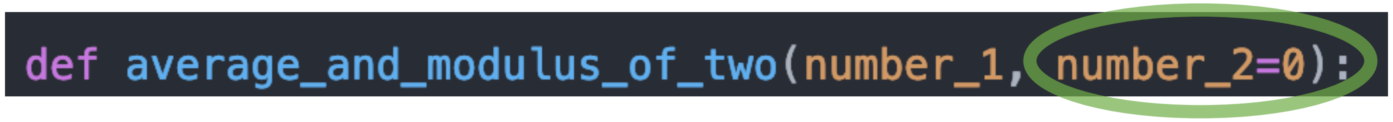


```
def average_and_modulus_of_two(number_1, number_2=0):
```

Python Functions

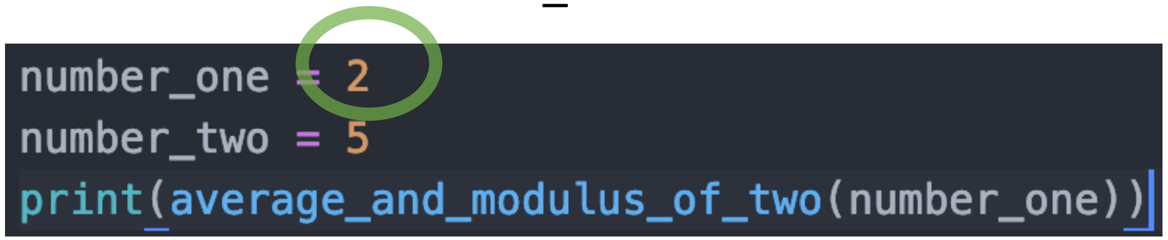
To overcome the error we can set a default value to number_2

In this instance we set it to “0” - zero



```python
def average_and_modulus_of_two(number_1, number_2=0):
```

If ‘number_one’ is set to 2

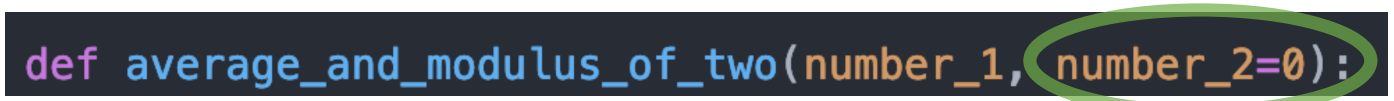


```python
number_one = 2
number_two = 5
print(average_and_modulus_of_two(number_one))
```

Python Functions

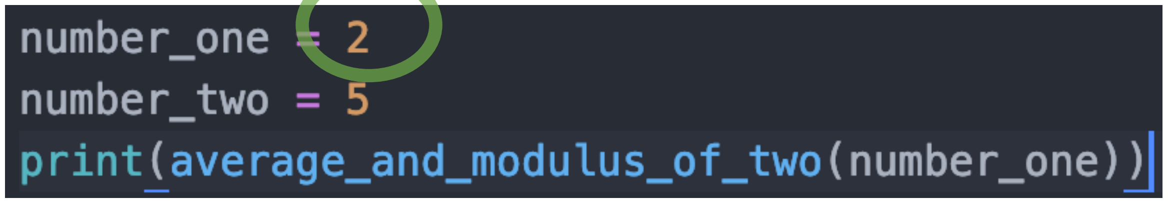
To overcome the error we can set a default value to number_2

In this instance we set it to “0” - zero



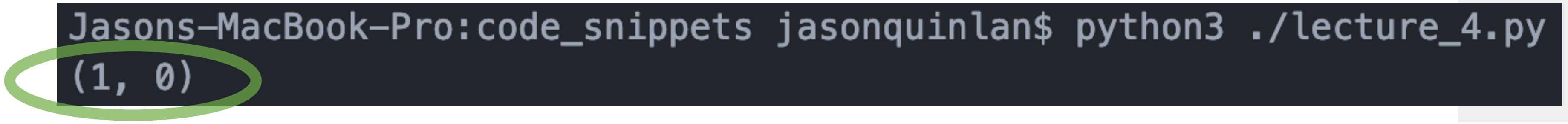
```
def average_and_modulus_of_two(number_1, number_2=0):
```

If ‘number_one’ is set to 2



```
number_one = 2
number_two = 5
print(average_and_modulus_of_two(number_one))
```

No error



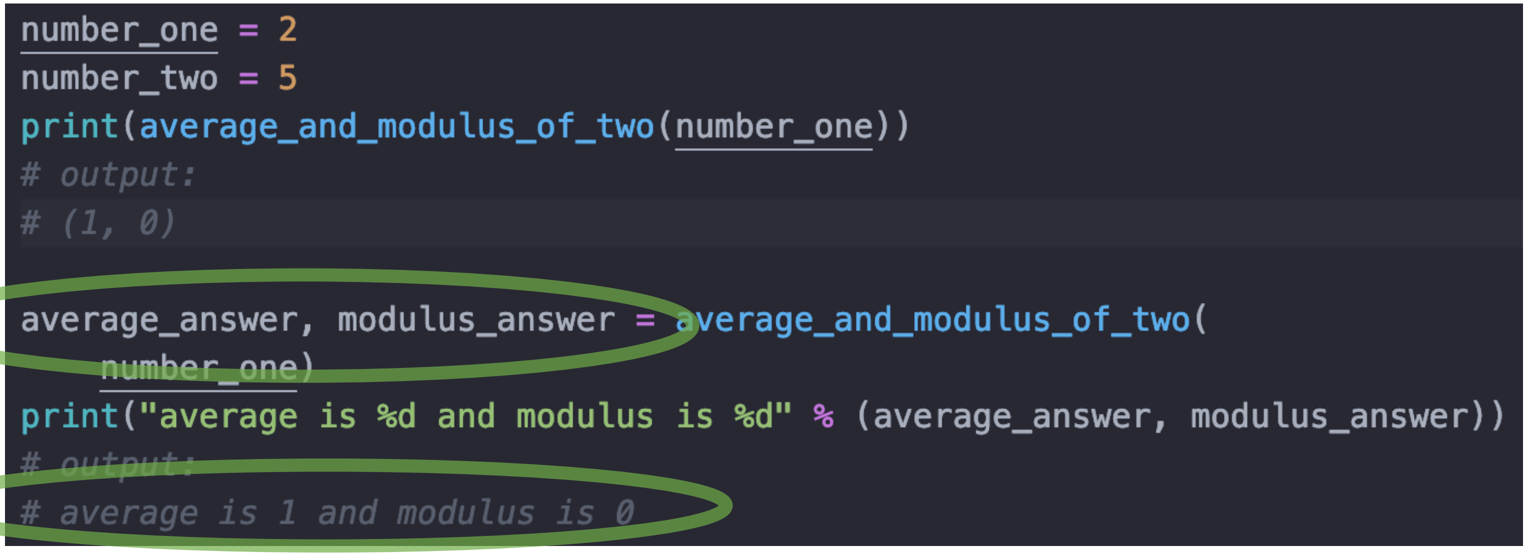
```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_4.py
(1, 0)
```

And as modulus is zero, it’s an even number

Python Functions

If we save the returned values as variables

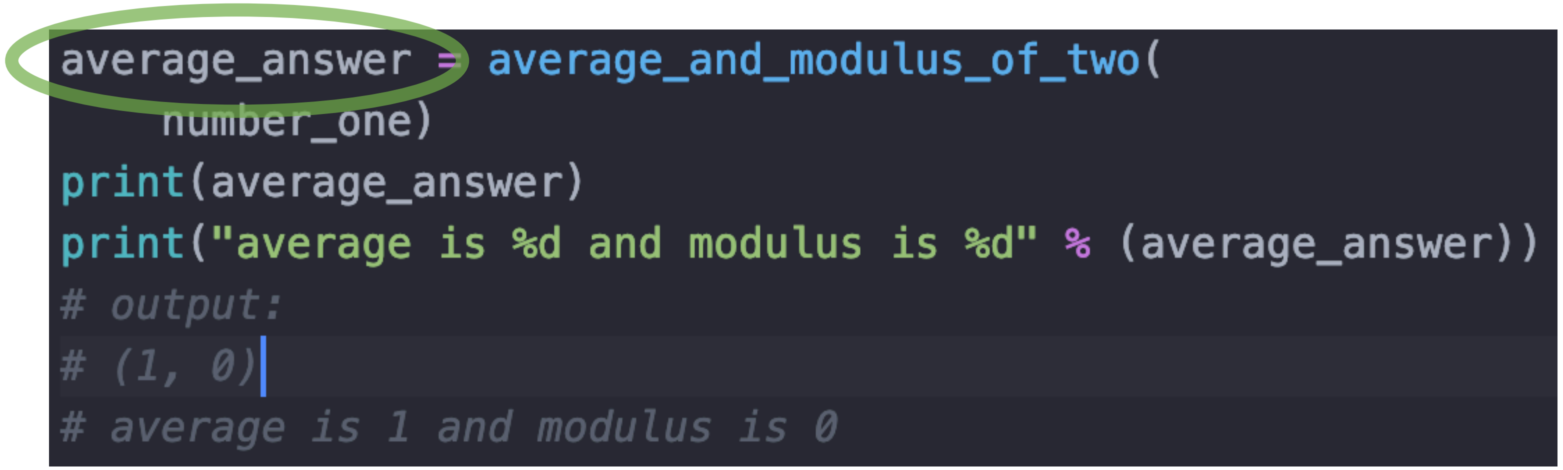
We get the same result



```
number_one = 2
number_two = 5
print(average_and_modulus_of_two(number_one))
# output:
# (1, 0)

average_answer, modulus_answer = average_and_modulus_of_two(
    number_one)
print("average is %d and modulus is %d" % (average_answer, modulus_answer))
# output:
# average is 1 and modulus is 0
```

Python Functions

Finally, if we assign the returned "tuple" as a single variable

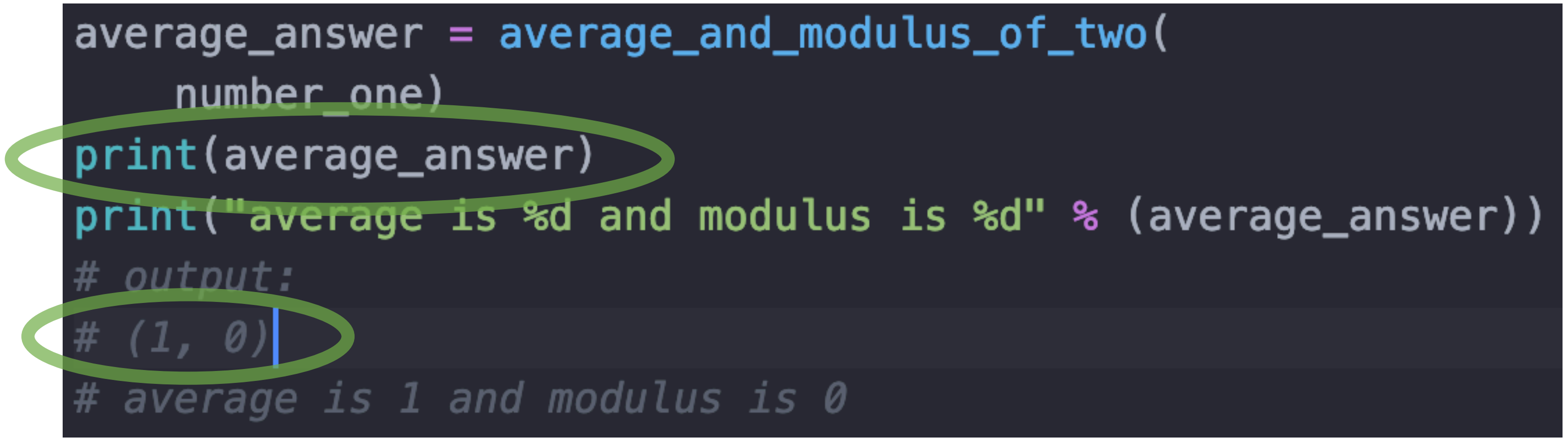


```
average_answer = average_and_modulus_of_two(
    number_one)
print(average_answer)
print("average is %d and modulus is %d" % (average_answer))
# output:
# (1, 0)
# average is 1 and modulus is 0
```

Python Functions

Finally, if we assign the returned "tuple" as a single variable

If we print the "tuple" we can see the returned values (1,0)



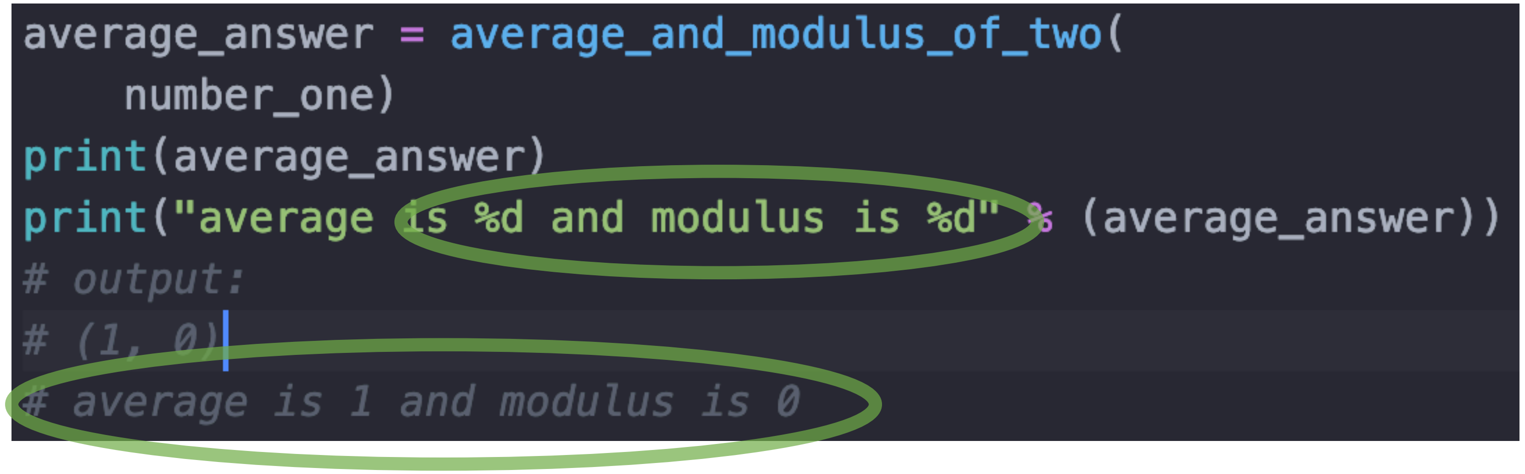
```
average_answer = average_and_modulus_of_two(
    number_one)
print(average_answer)
print("average is %d and modulus is %d" % (average_answer))
# output:
# (1, 0)
# average is 1 and modulus is 0
```

Python Functions

Finally, if we assign the returned "tuple" as a single variable

If we print the "tuple" we can see the returned values (1,0)

If we want to access the values within the "tuple"
we can use the string format %d



```
average_answer = average_and_modulus_of_two(
    number_one)
print(average_answer)
print("average is %d and modulus is %d" % (average_answer))
# output:
# (1, 0)
# average is 1 and modulus is 0
```

Python Functions Recap 1

- We saw how we can take similar code and create a function
- We saw how to define the function:
  - def function_name(function_parameter):
- We saw how we indent code within the function
  - Known as a block of statements
  - So Python knows which code belongs in the function
- We saw how to return a value
  - Back to the line of code that called the function
  - And allocate the returning value to a variable
- We used id() to find the unique integer for variable values
  - If variables have the same value, they point to the same object and have the same id
  - Calling a function with a parameter, allocates the same id to the value of both the function parameter and the variable in the function call

Python Functions Recap 2

- We wrote an “average_of_two” function
  - That calculates the average of two numbers
  - And returns said average
- We add a docstring comment, which is viewable in an IDE
- We noted that our returned value was a float
  - Generating a value with a decimal place
- We looked at Python Operators
  - (=), (+), (/), (*), (-), (**), (%) and (//)
  - We noted that the operators are mutable
    - Change their operation based on the data type they are working with
  - They have precedence (similar to BOMDAS)

Python Functions Recap 3

- We added prints to our "average_of_two" function
  - That prints the value of the two input numbers
- We noted we could set the value of the function parameter directly in the function call
  - Allows us to change the order of the inputs
- We created a new function "average_and_modulus_of_two"
  - Which takes the same inputs as the "average_of_two" function
  - But returns two values
- Modification to the function consist of:
  - We changed the float division to integer division
  - We added modulus to get the remainder
- The result of the function call returned a:
  - Tuple when printed directly
  - but two values when assigned directly to two variables
- We looked at setting a default value for one parameters
  - Allowing us to call the function with only one parameter

Using Python Functions

We wrote an “average_of_two” function

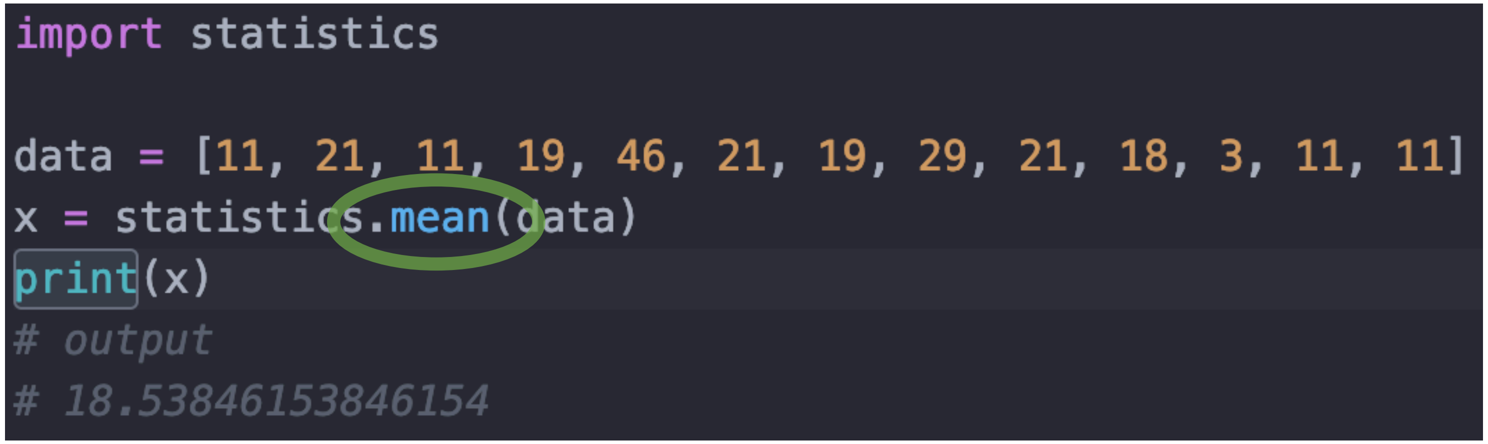
But Python has its own library of math functions which are stored in a module called statistics

Using Python Functions

We wrote an “average_of_two” function

But Python has its own library of math functions which are stored in a module called statistics

Python provides a “mean()” function which returns the average of the numbers passed in as a list parameter



```python
import statistics

data = [11, 21, 11, 19, 46, 21, 19, 29, 21, 18, 3, 11, 11]
x = statistics.mean(data)
print(x)
# output
# 18.53846153846154
```

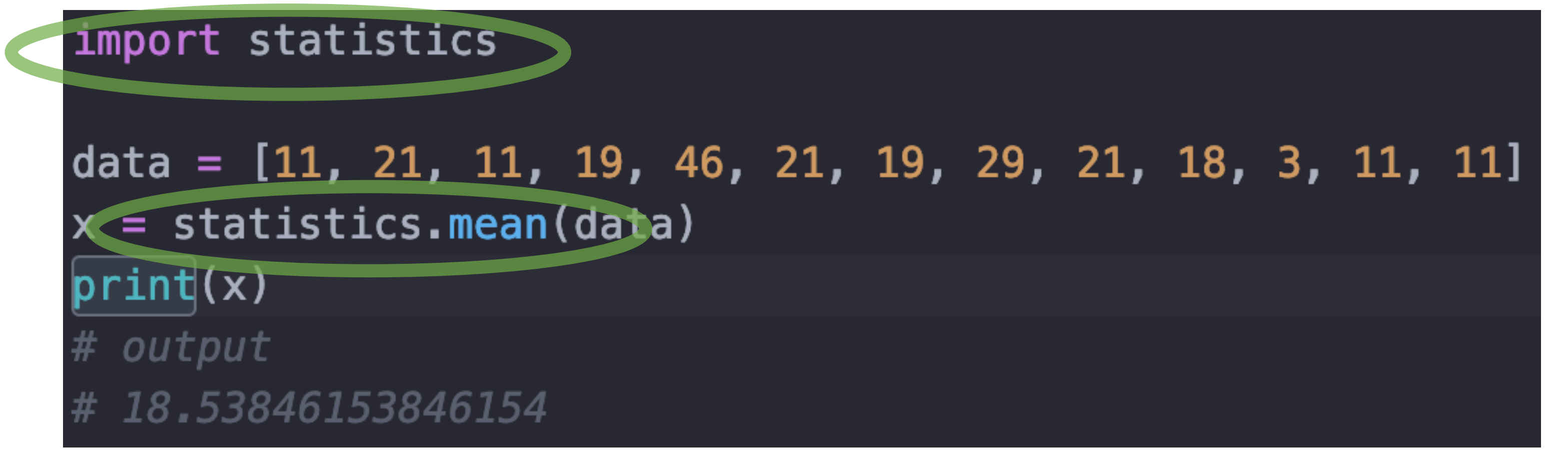
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Using Python Functions

We wrote an "average_of_two" function

But Python has its own library of math functions which are stored in a module called statistics

But to use the "mean()" function, we need to import the statistics module into our python file



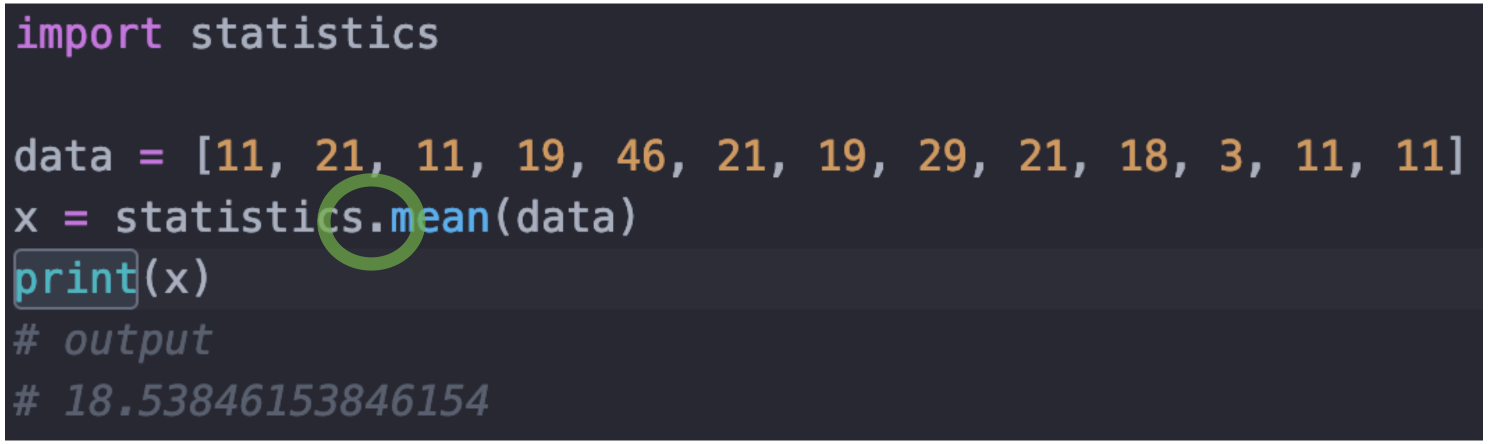
```
import statistics

data = [11, 21, 11, 19, 46, 21, 19, 29, 21, 18, 3, 11, 11]
x = statistics.mean(data)
print(x)
# output
# 18.53846153846154
```

Using Python Functions

We wrote an "average_of_two" function

But Python has its own library of math functions which are stored in a module called statistics

But to use the "mean()" function, we need to import the statistics module into our python file



```
import statistics

data = [11, 21, 11, 19, 46, 21, 19, 29, 21, 18, 3, 11, 11]
x = statistics.mean(data)
print(x)
# output
# 18.538461538461 54
```

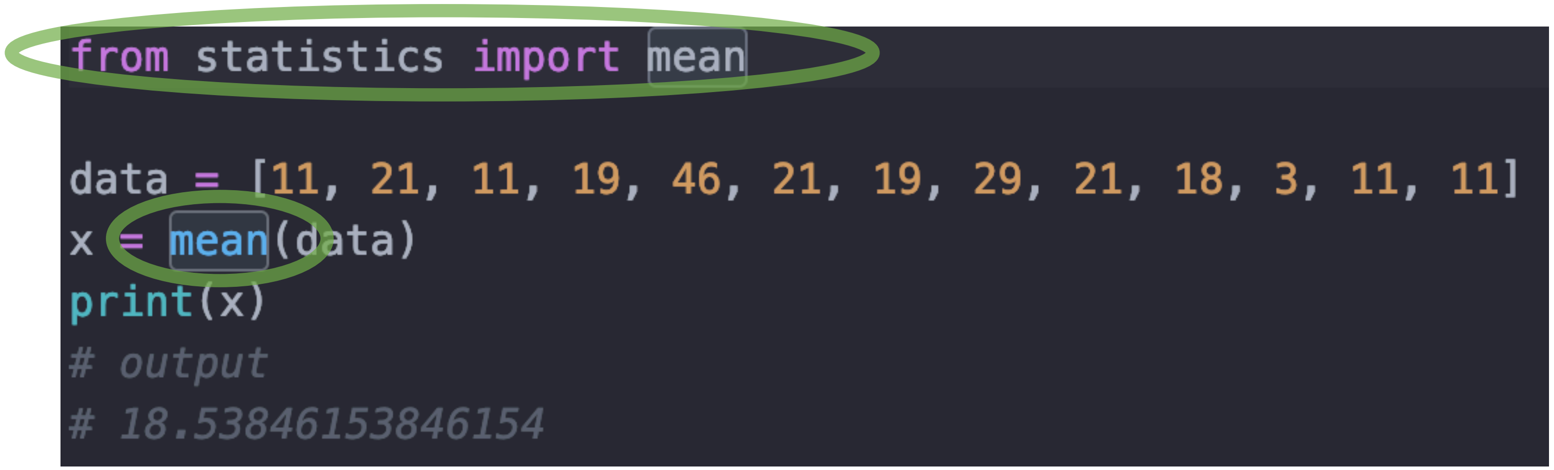
Mean is called using the dot (.) operator

Using Python Functions

By using import statistics we import all functions in Pythons statistics module

These include mean(), median(), mode(), stdev() and variance()

If we only want to use mean(), we can modify our import



```
from statistics import mean

data = [11, 21, 11, 19, 46, 21, 19, 29, 21, 18, 3, 11, 11]
x = mean(data)
print(x)
# output
# 18.53846153846154
```

Python Print() Function

Now we know how to define a function and call the function

We can modify inputs parameters (setting default values) and generate multiple return values

And we can import functions from Pythons in-built libraries

Let’s look at print() again

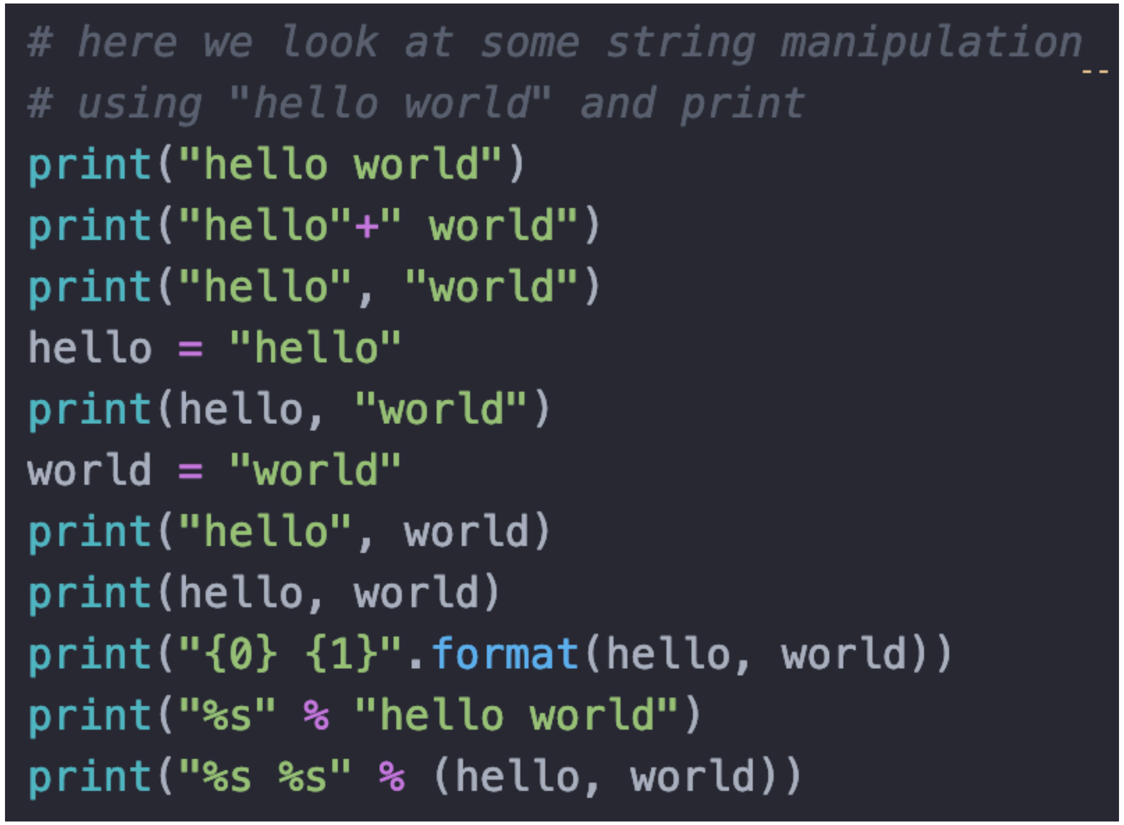
We know print() takes a string and prints to the screen

Let’s look at how we can modify the string we give to print()

String Manipulation

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

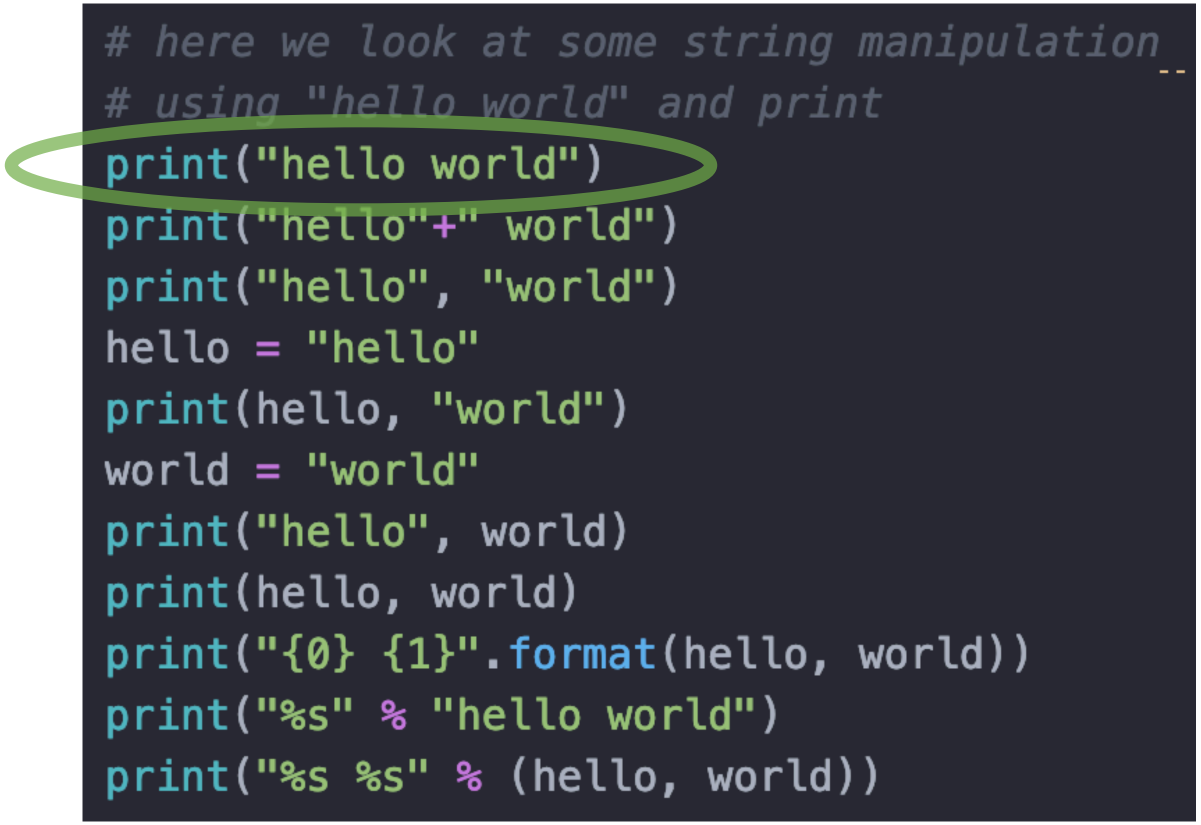
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

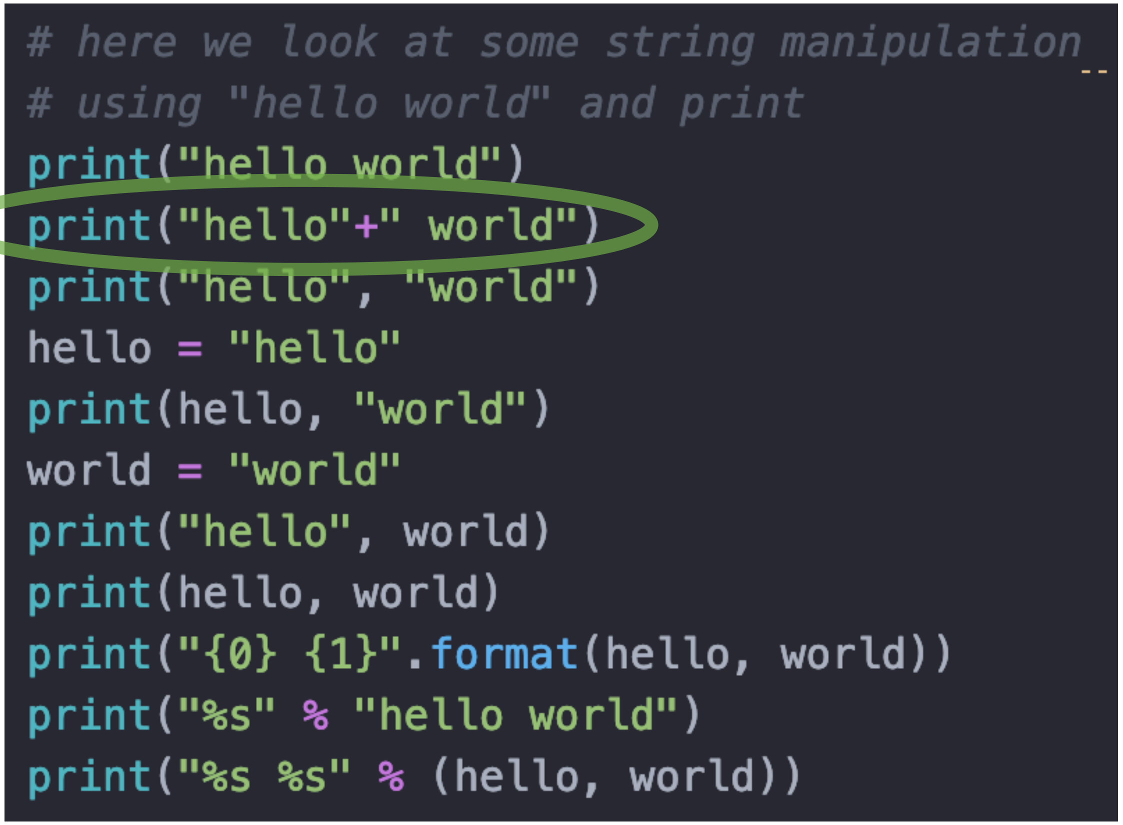
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello world”) – print a string literal

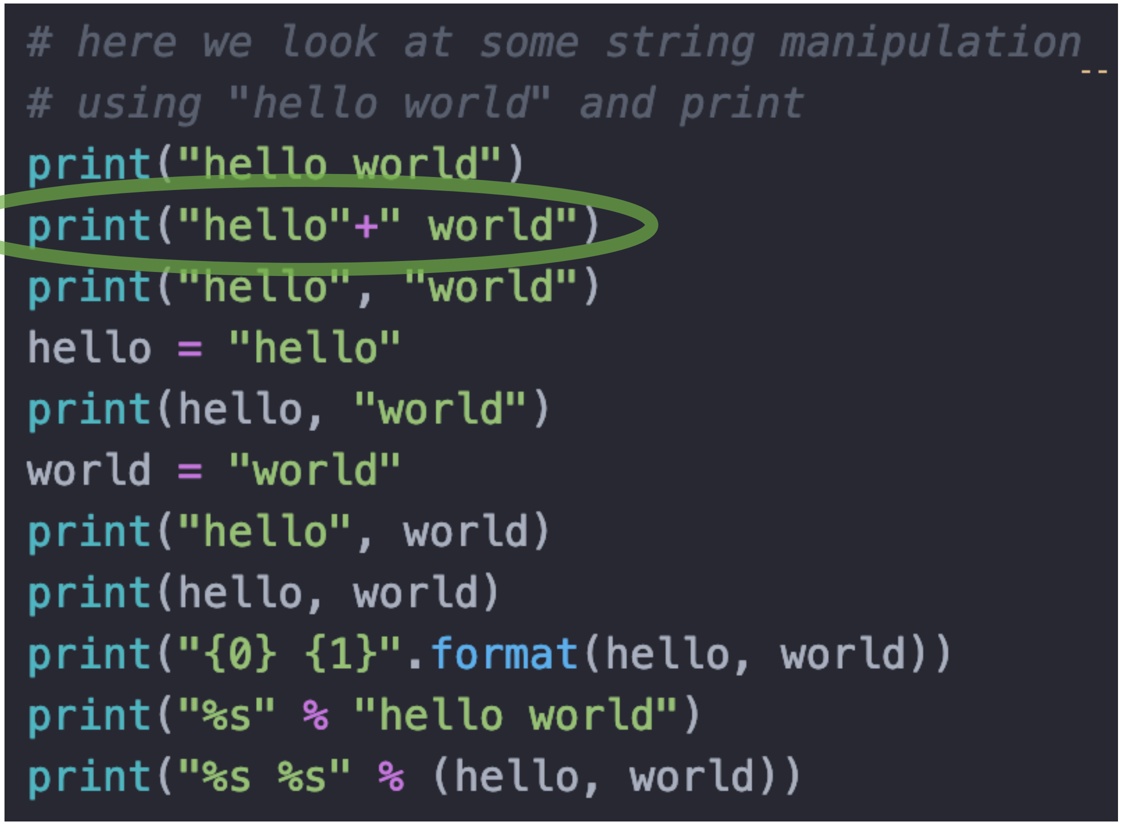
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello”+” world”) – concatenate two string together

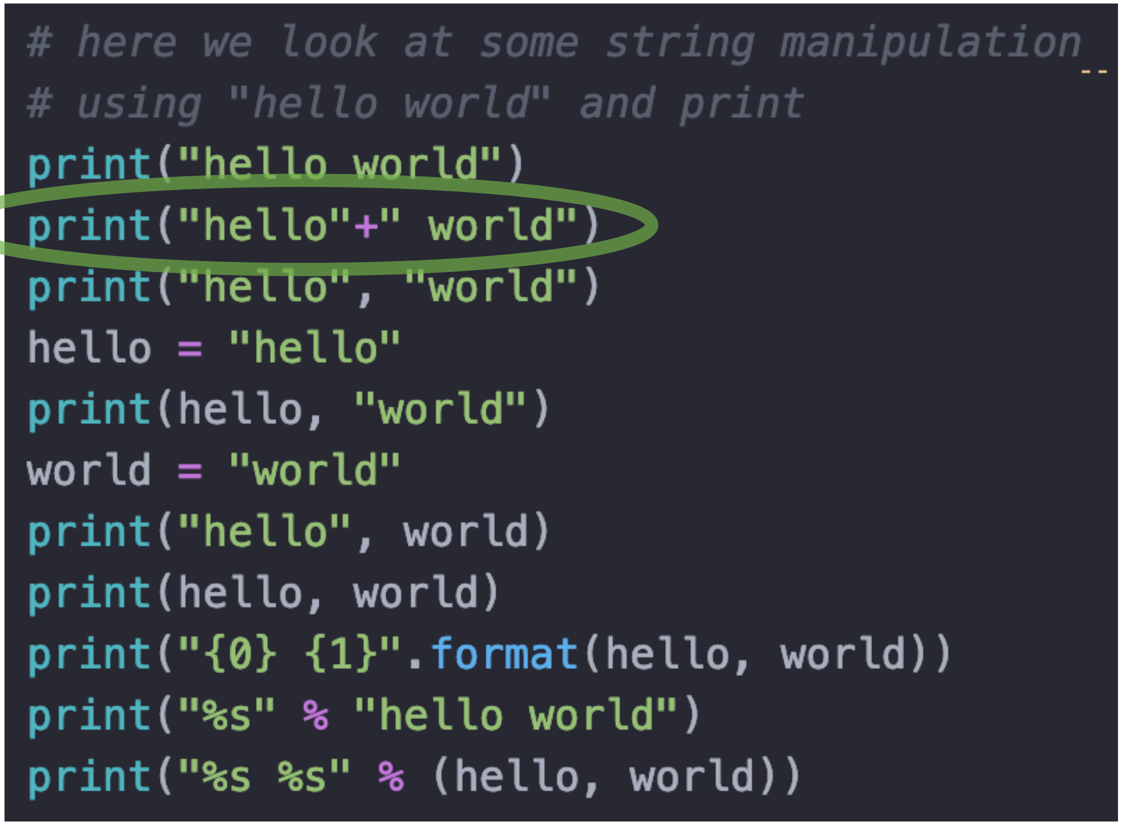
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello”+” world”) – concatenate two string together
Note: you need to keep the extra space
at the start of “ world”

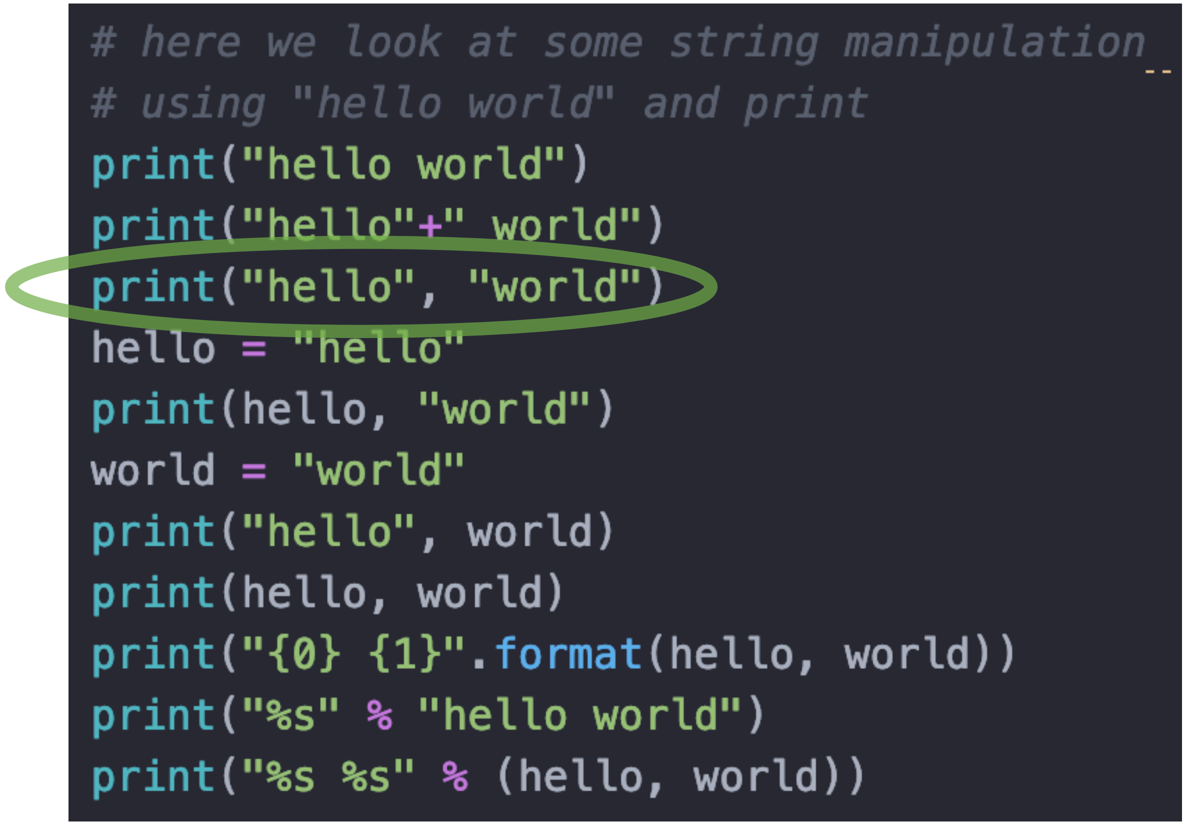
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello”+” world”) – concatenate two string together
Note: you need to keep the extra space
at the start of “ world”

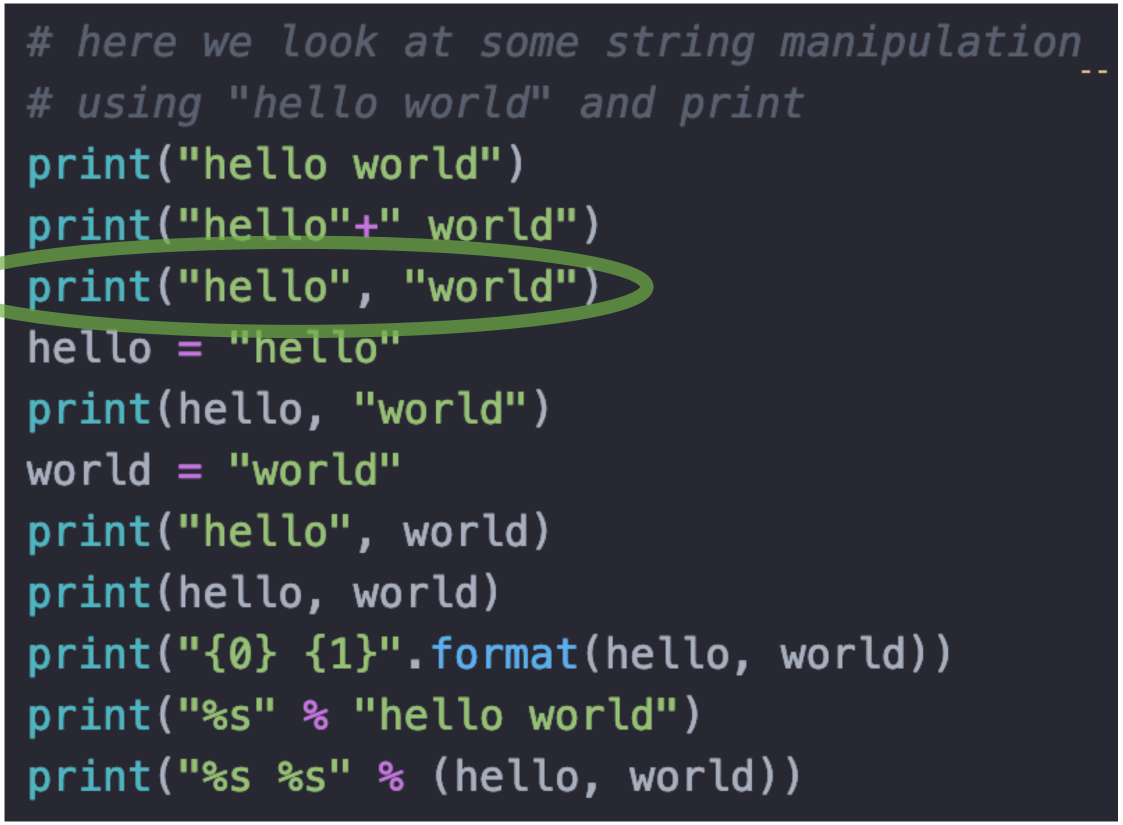
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello”, “world”) – concatenate two string together

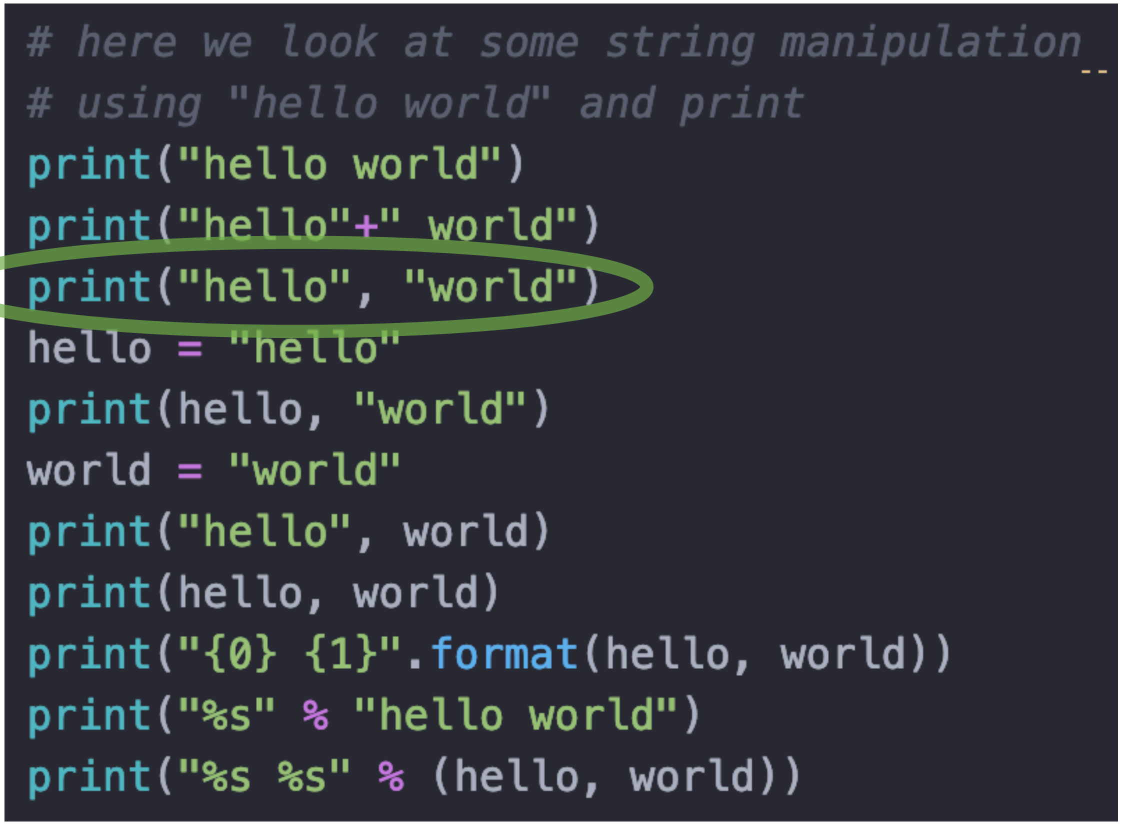
String Manipulation



```
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello”, “world”) – concatenate two string together
No need for the extra space at the start of “world”

String Manipulation



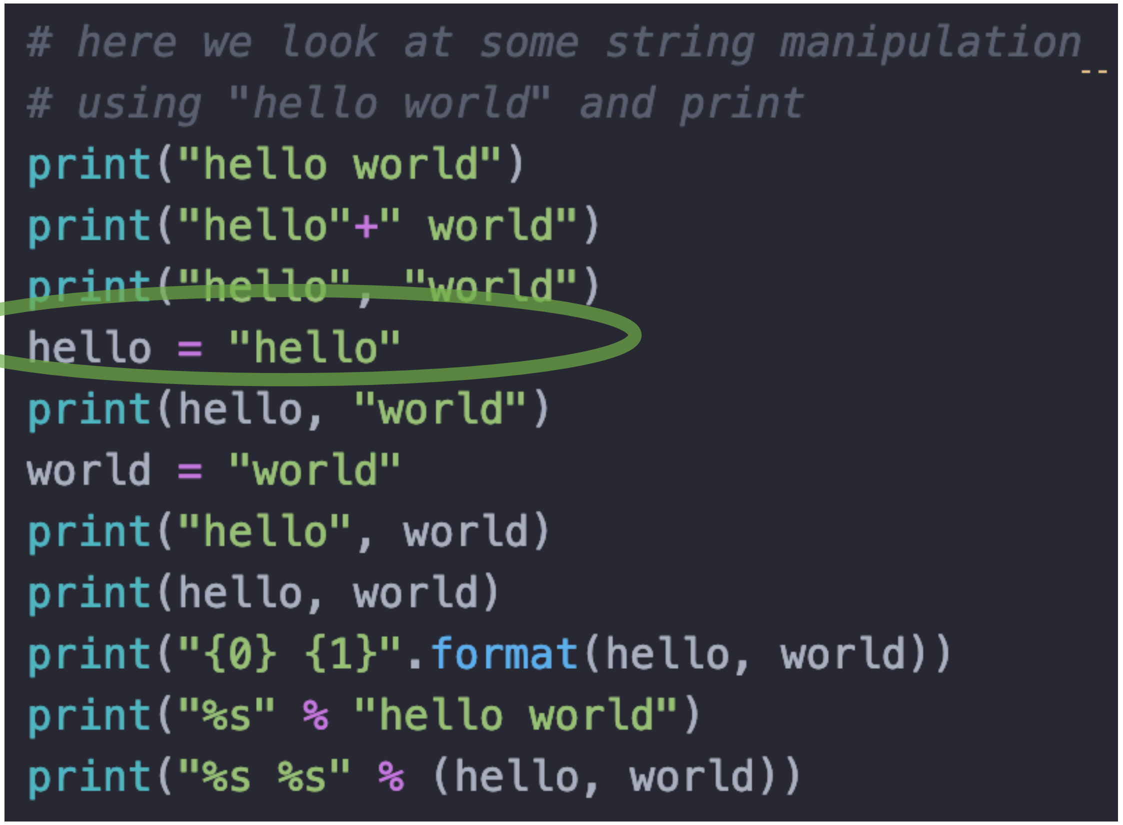
```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello”, “world”) – concatenate two string together

No need for the extra space at the start of “world”

This will be automatically added by print()

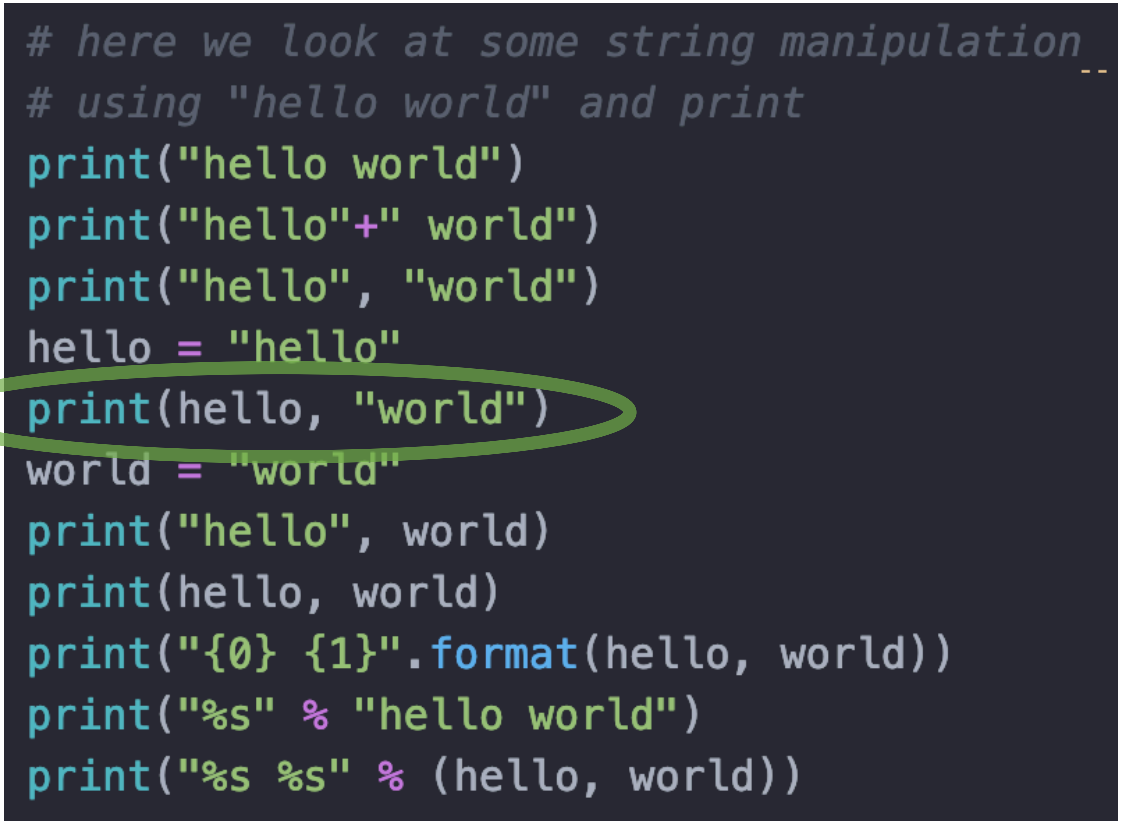
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello world”) – create a variable for “hello”

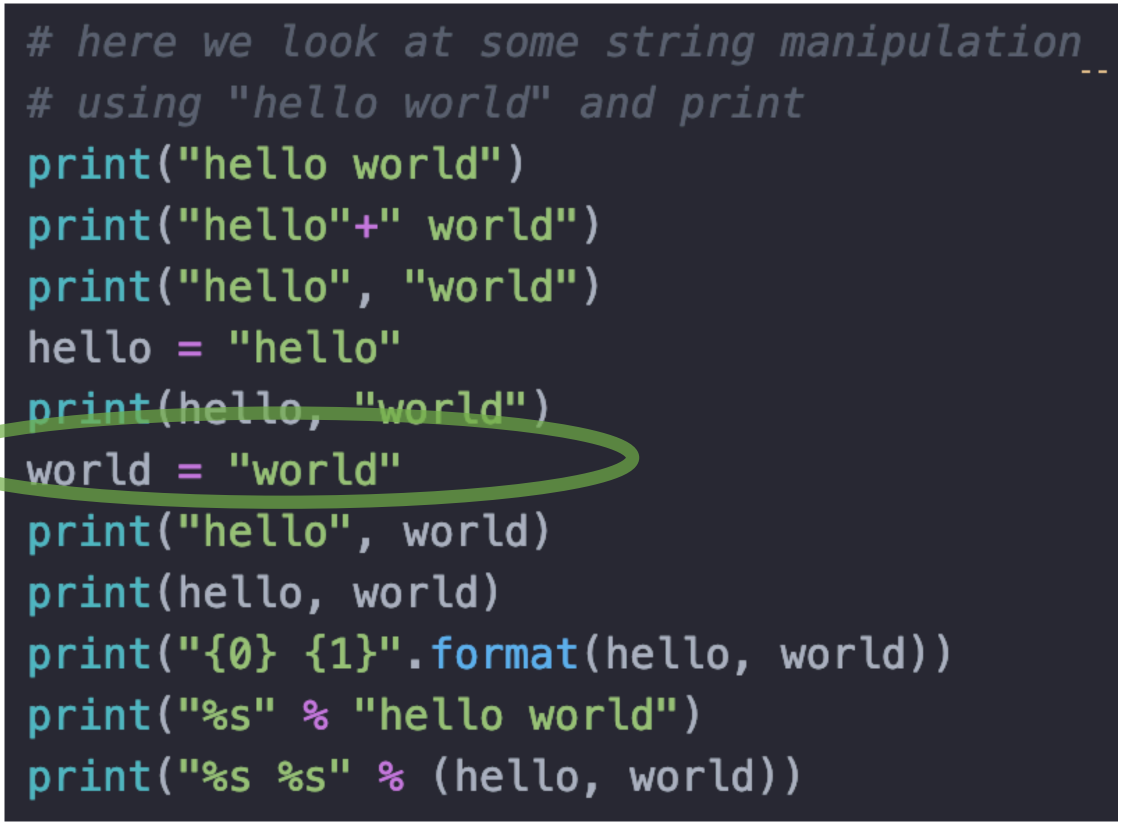
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print("hello world") – create a variable for "hello"
And use the variable in print()

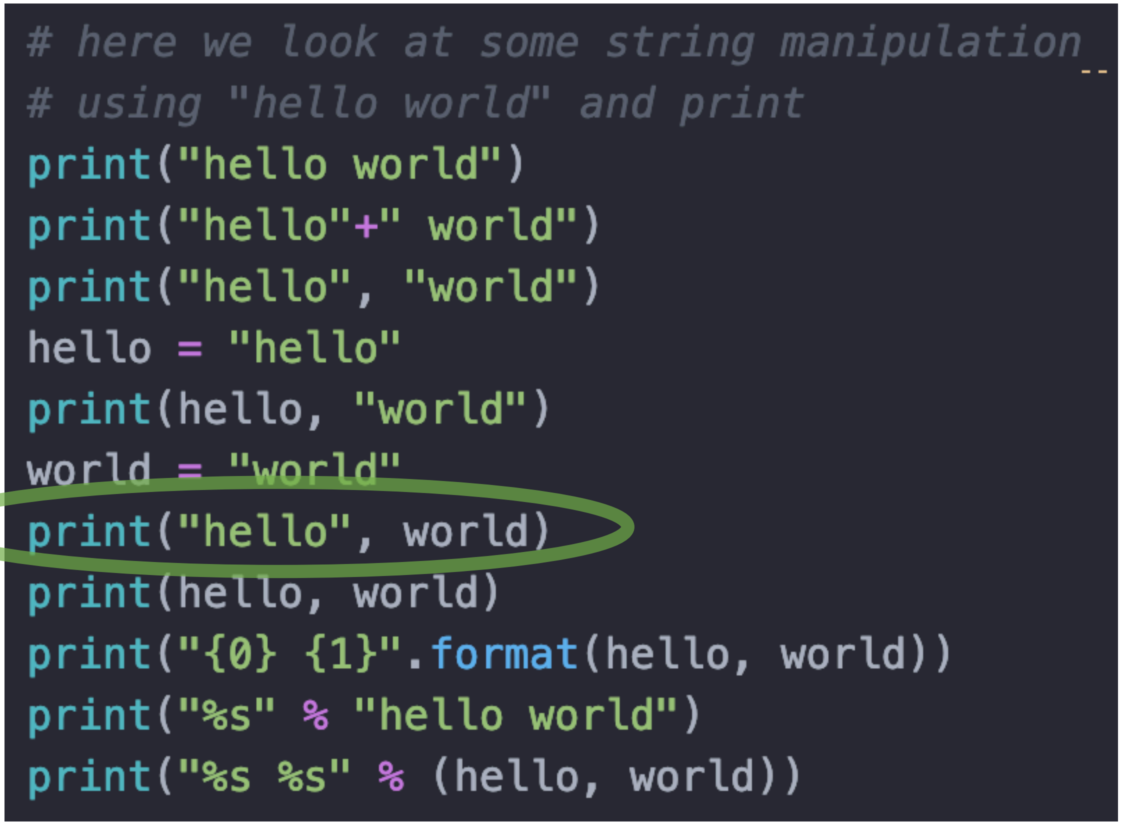
String Manipulation



```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello world”) – create a variable for “world”

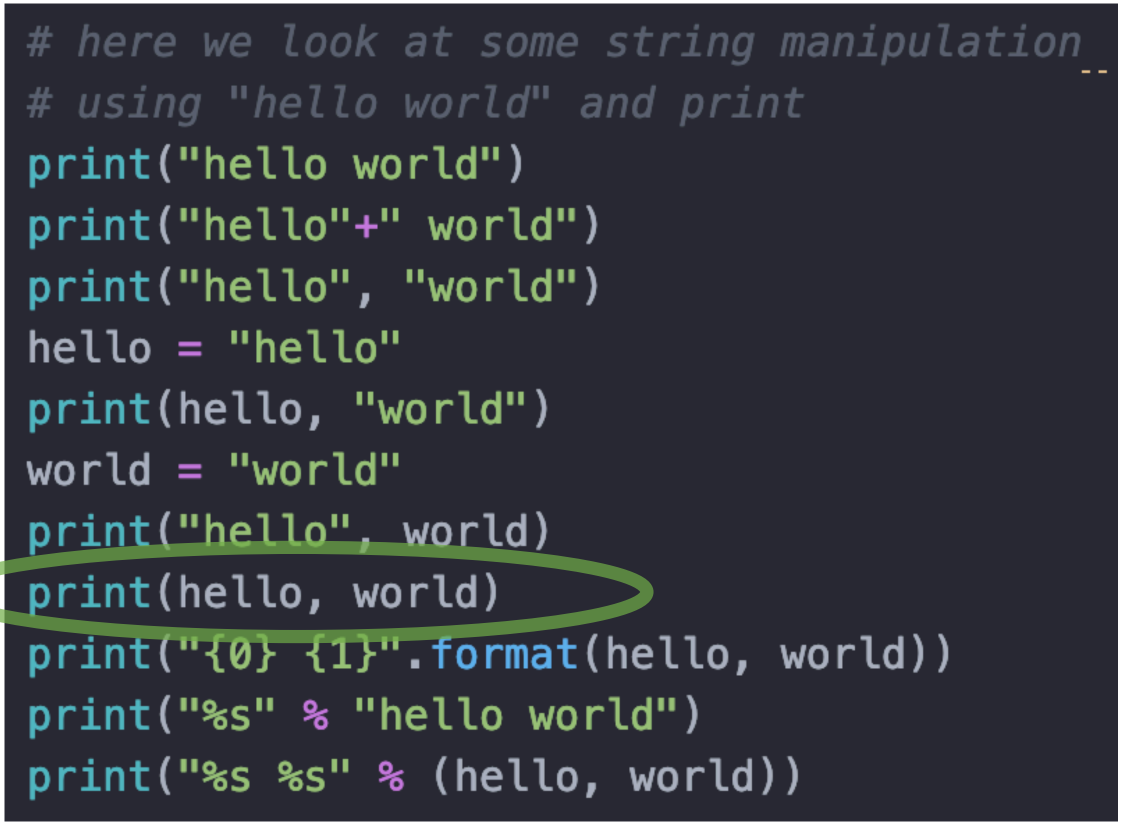
String Manipulation



```
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print("hello world") – create a variable for "world"
And use the variable in print()

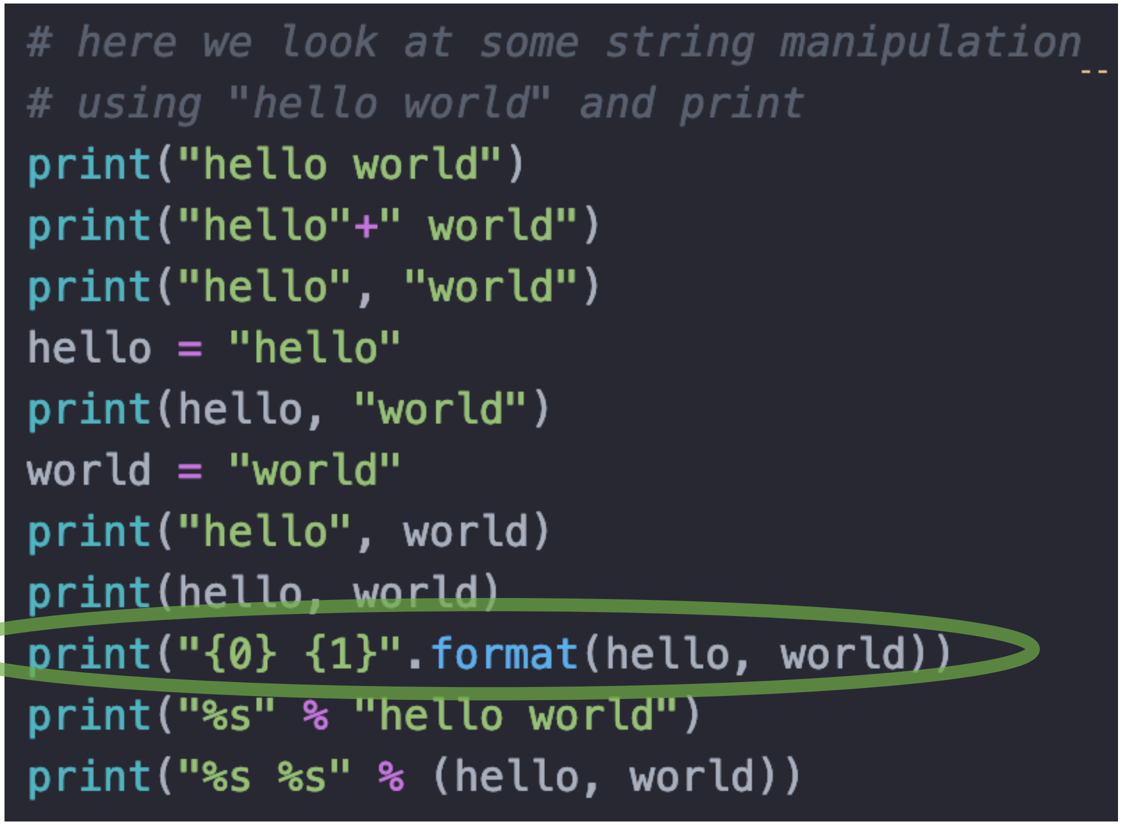
String Manipulation



```
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print("hello world") – use both the hello and world variables in print()

String Manipulation



```
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print("hello world") – use both the hello and world variables and pass them as parameters to the format() function of string

String Functions

```
hello = "hello”
world = "world"
print("{0} {1}".format(hello, world))
```

"{0} {1}” is a string object

For string objects, Python has created a range of functions to perform frequency occurring task (related to string).

For example, if you want to capitalize the first letter of a string, you can use the capitalize() function.

String Functions

Python String capitalize()	Converts first character to Capital Letter
Python String endswith()	Checks if String Ends with the Specified Suffix
Python String find()	Returns the index of first occurrence of substring
Python String format()	formats string into nicer output
Python String index()	Returns Index of Substring
Python String isalnum()	Checks Alphanumeric Character
Python String isalpha()	Checks if All Characters are Alphabets
Python String isdecimal()	Checks Decimal Characters
Python String isdigit()	Checks Digit Characters
Python String islower()	Checks if all Alphabets in a String are Lowercase
Python String isnumeric()	Checks Numeric Characters
Python String isupper()	returns if all characters are uppercase characters
Python String lower()	returns lowercased string
Python String upper()	returns uppercased string
Python String lstrip()	Removes Leading Characters
Python String rstrip()	Removes Trailing Characters
Python String strip()	Removes Both Leading and Trailing Characters
Python String partition()	Returns a Tuple

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Functions

Python String capitalize()	Converts first character to Capital Letter
Python String endswith()	Checks if String Ends with the Specified Suffix
Python String find()	Returns the index of first occurrence of substring
Python String format()	formats string into nicer output
Python String index()	Returns Index of Substring
Python String isalnum()	Checks Alphanumeric Character
Python String isalpha()	Checks if All Characters are Alphabets
Python String isdecimal()	Checks Decimal Characters
Python String isdigit()	Checks Digit Characters
Python String islower()	Checks if all Alphabets in a String are Lowercase
Python String isnumeric()	Checks Numeric Characters
Python String isupper()	returns if all characters are uppercase characters
Python String lower()	returns lowercased string
Python String upper()	returns uppercased string
Python String lstrip()	Removes Leading Characters
Python String rstrip()	Removes Trailing Characters
Python String strip()	Removes Both Leading and Trailing Characters
Python String partition()	Returns a Tuple

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Functions

```
hello = "hello”
world = "world"
print("{0} {1}".format(hello, world))
```

To call a specific function of any object we have created, we use the dot (.) operator

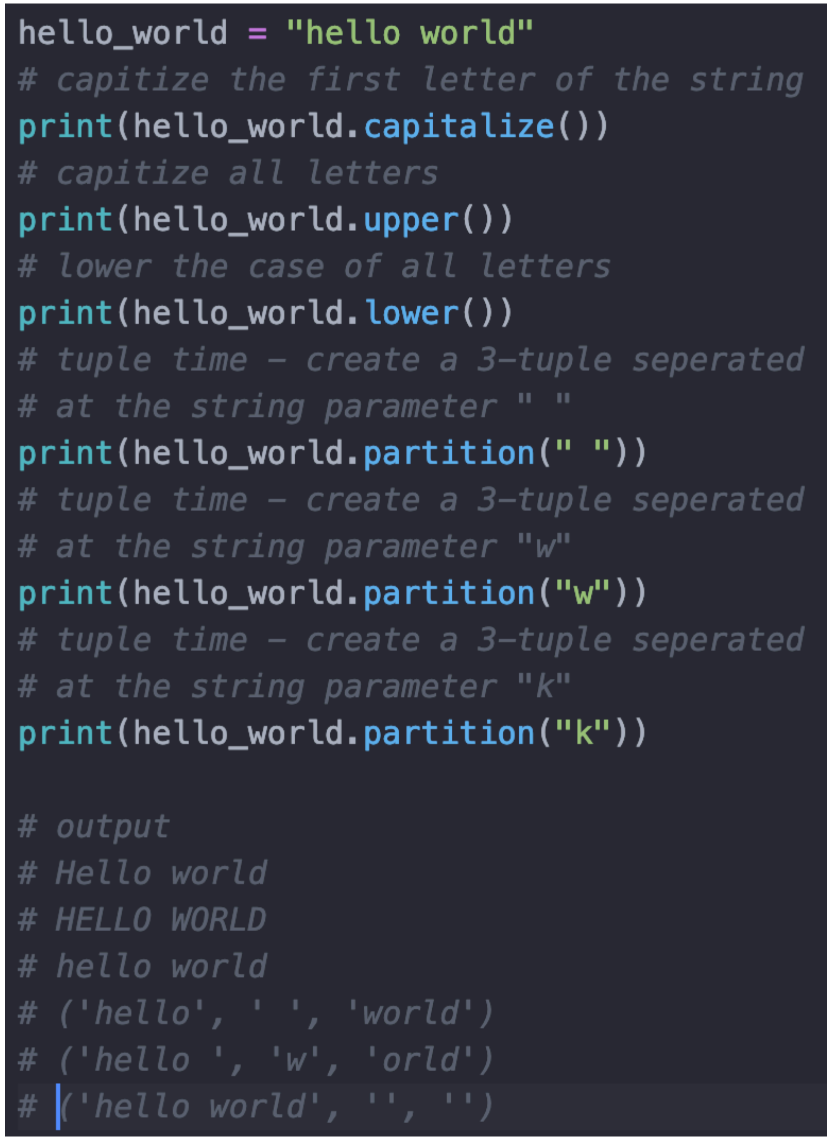
String Functions

```
hello = "hello”
world = "world"
print("{0} {1}".format(hello, world))
```

To call a specific function of any object we have created, we use the dot (.) operator

Let’s look at some examples

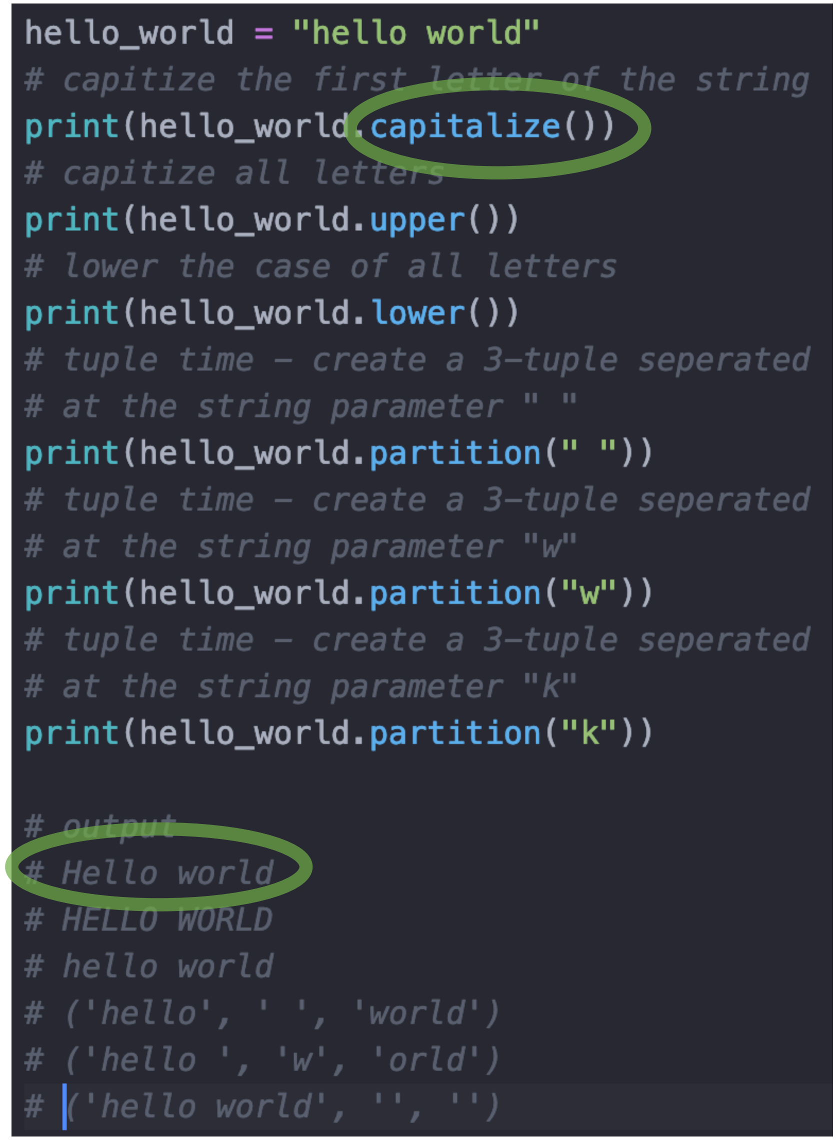
String Functions



```python
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

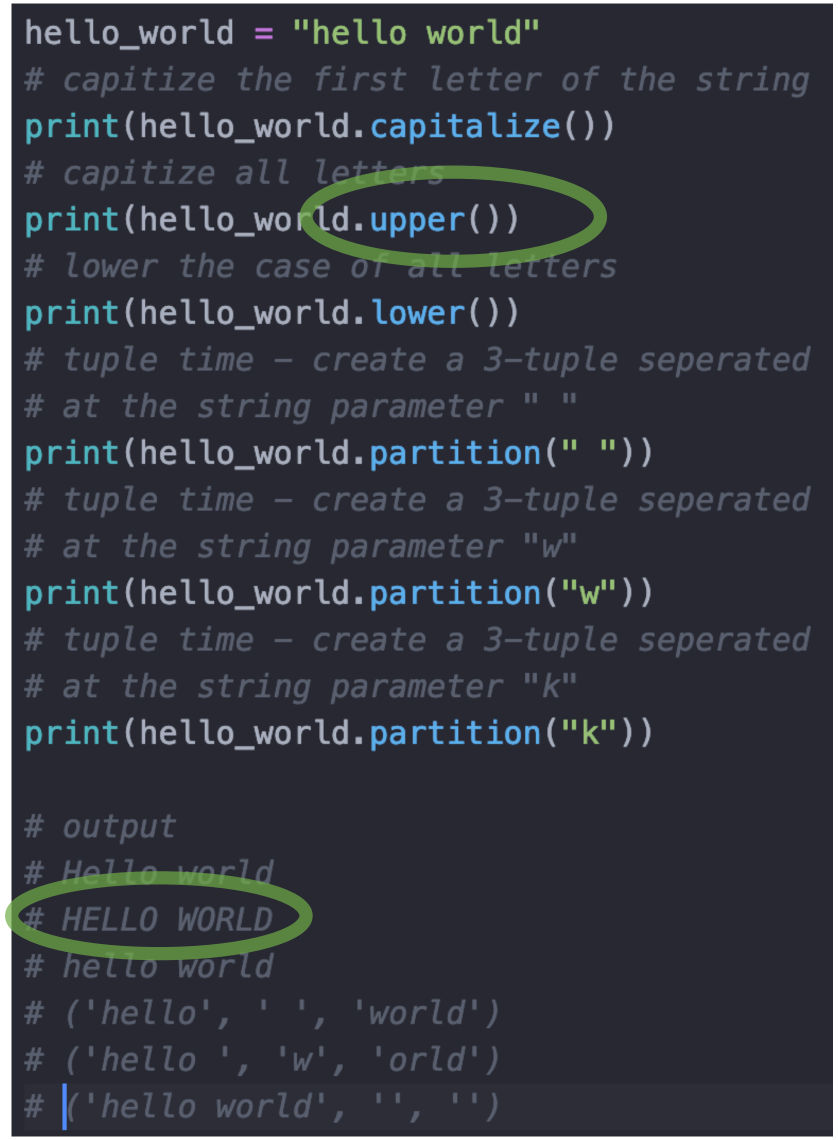
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

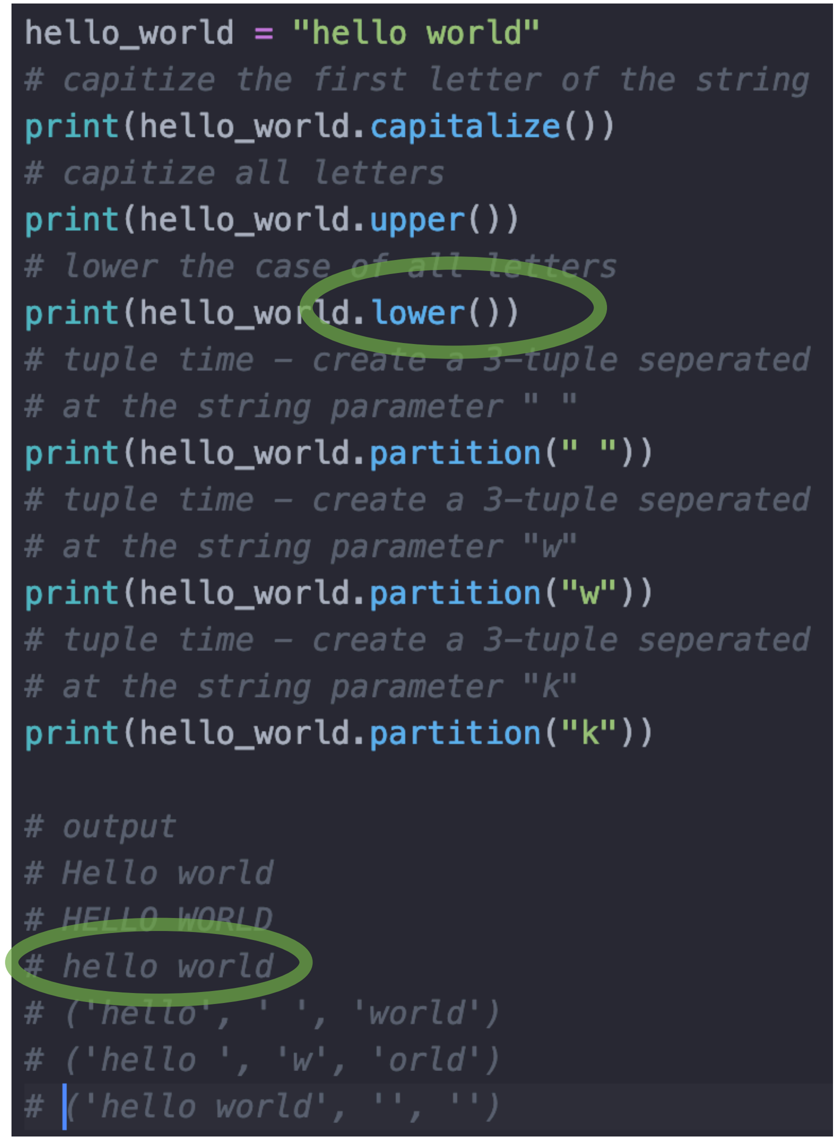
String Functions



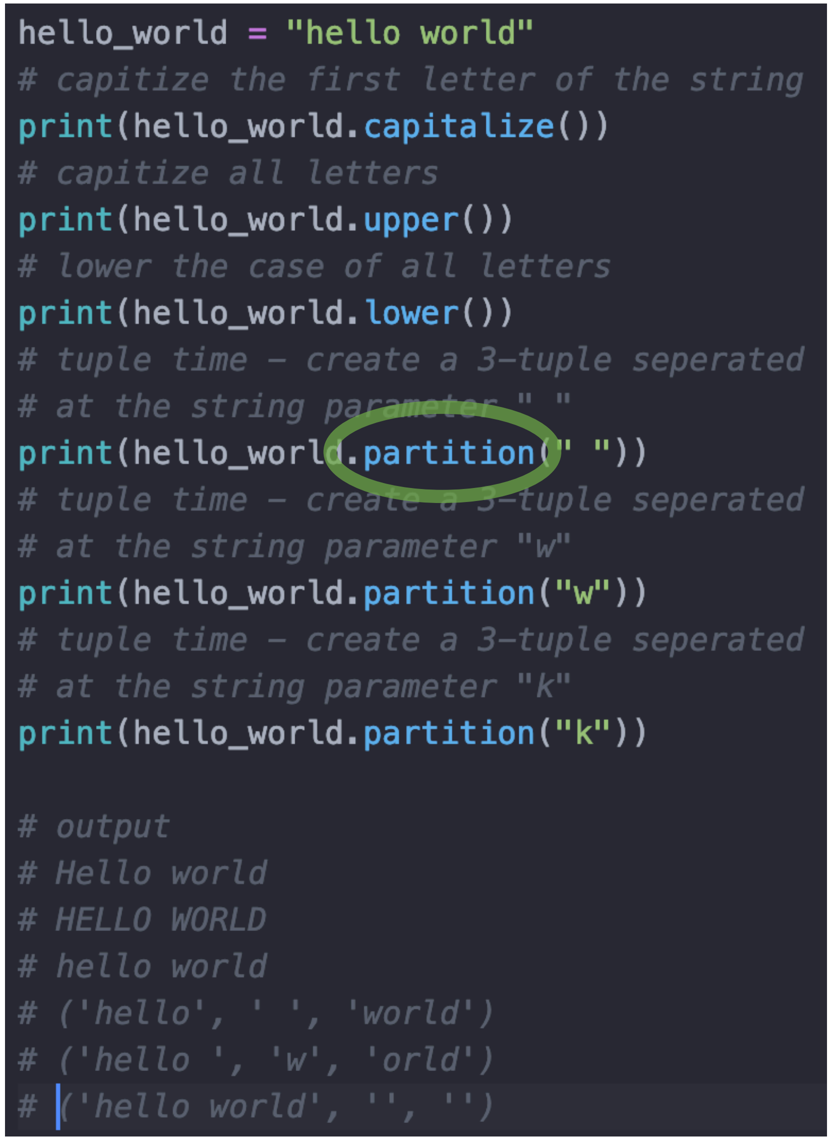
```python
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Functions

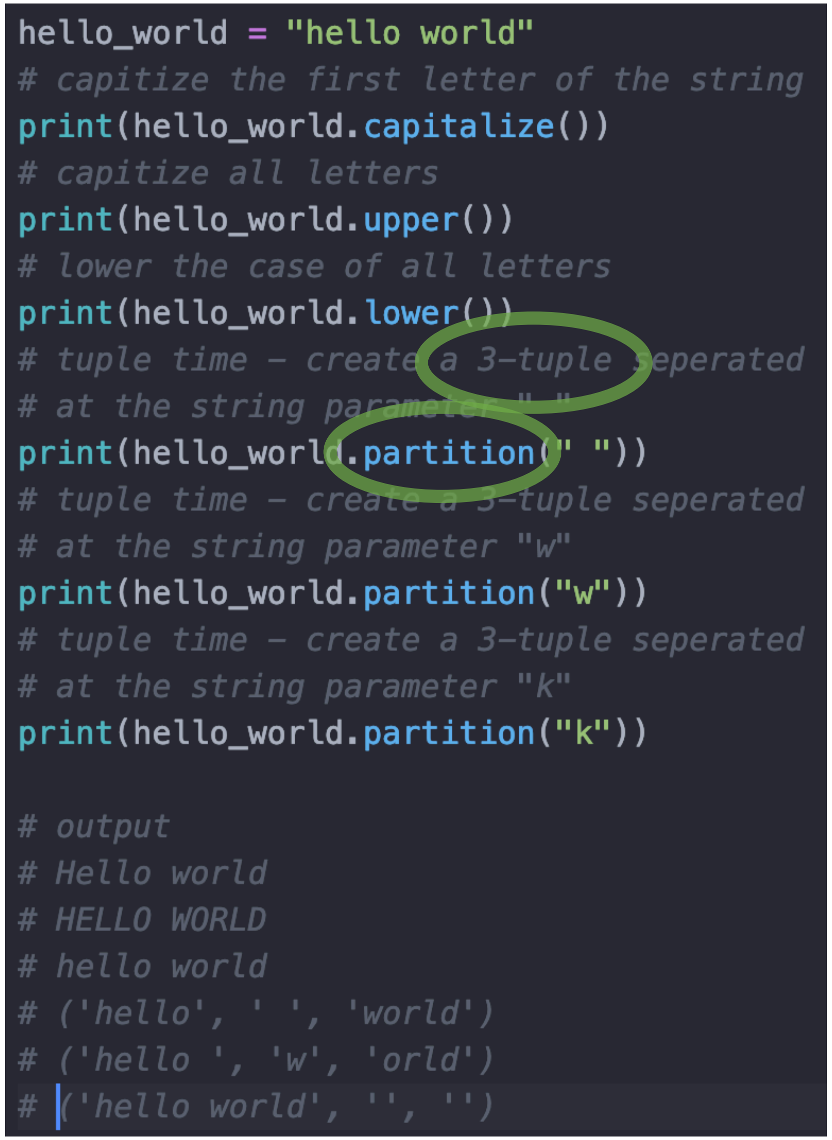

```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

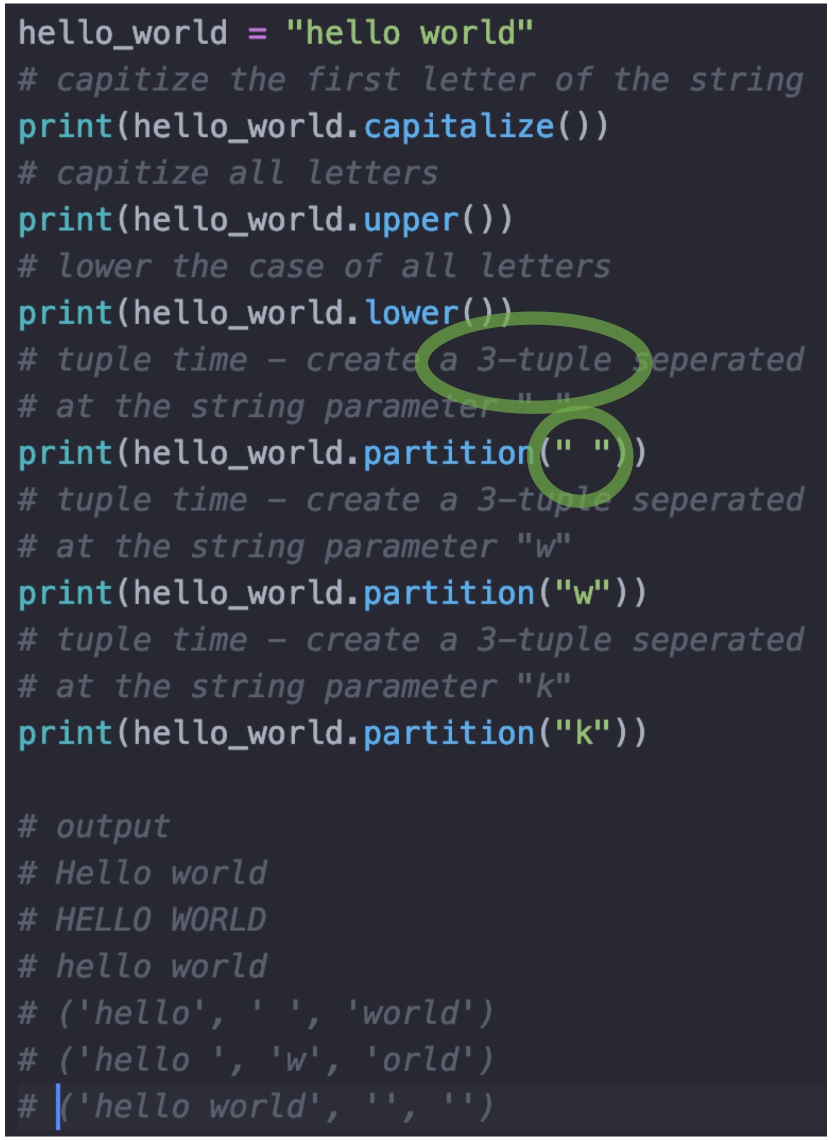
String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

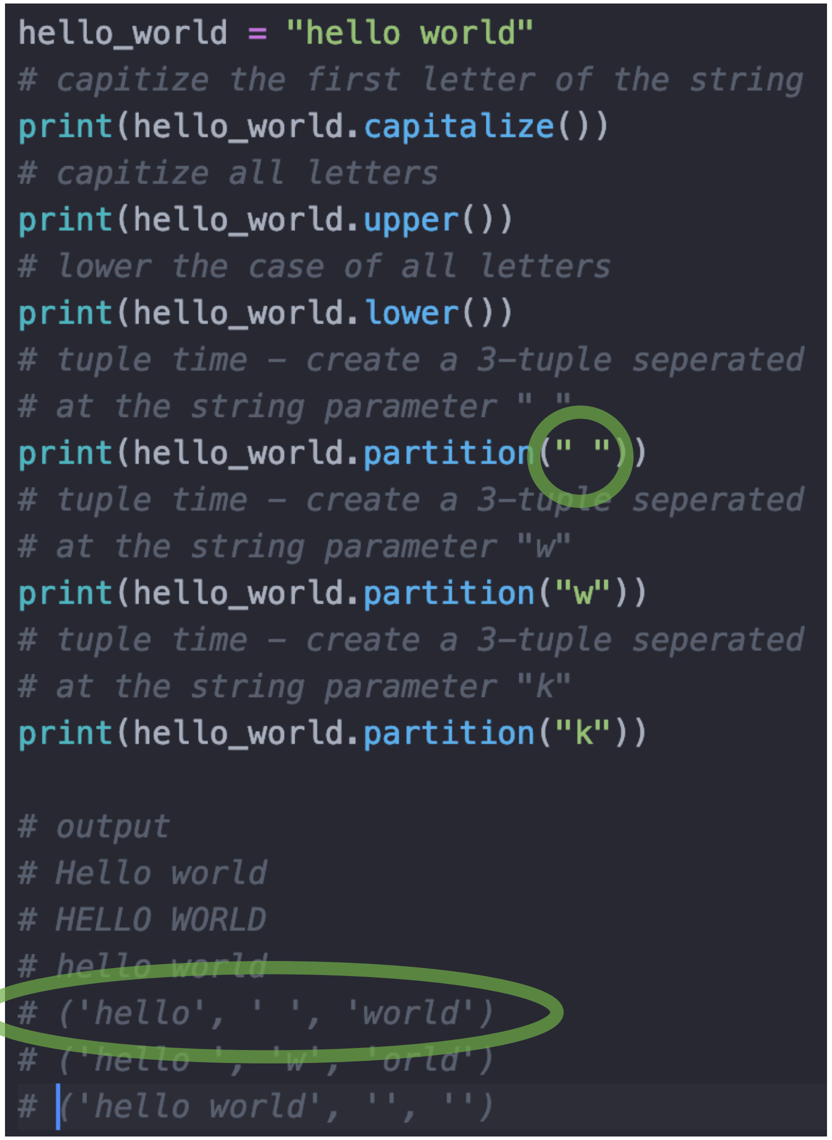
String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

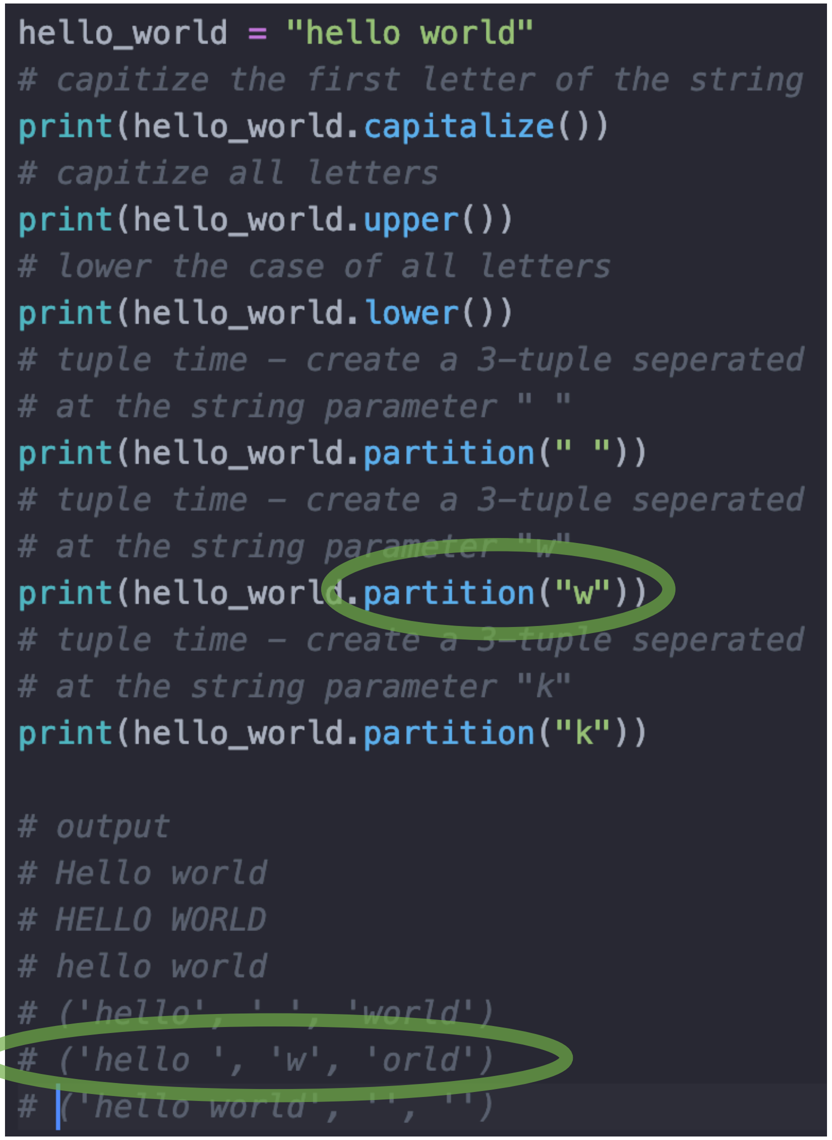
String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

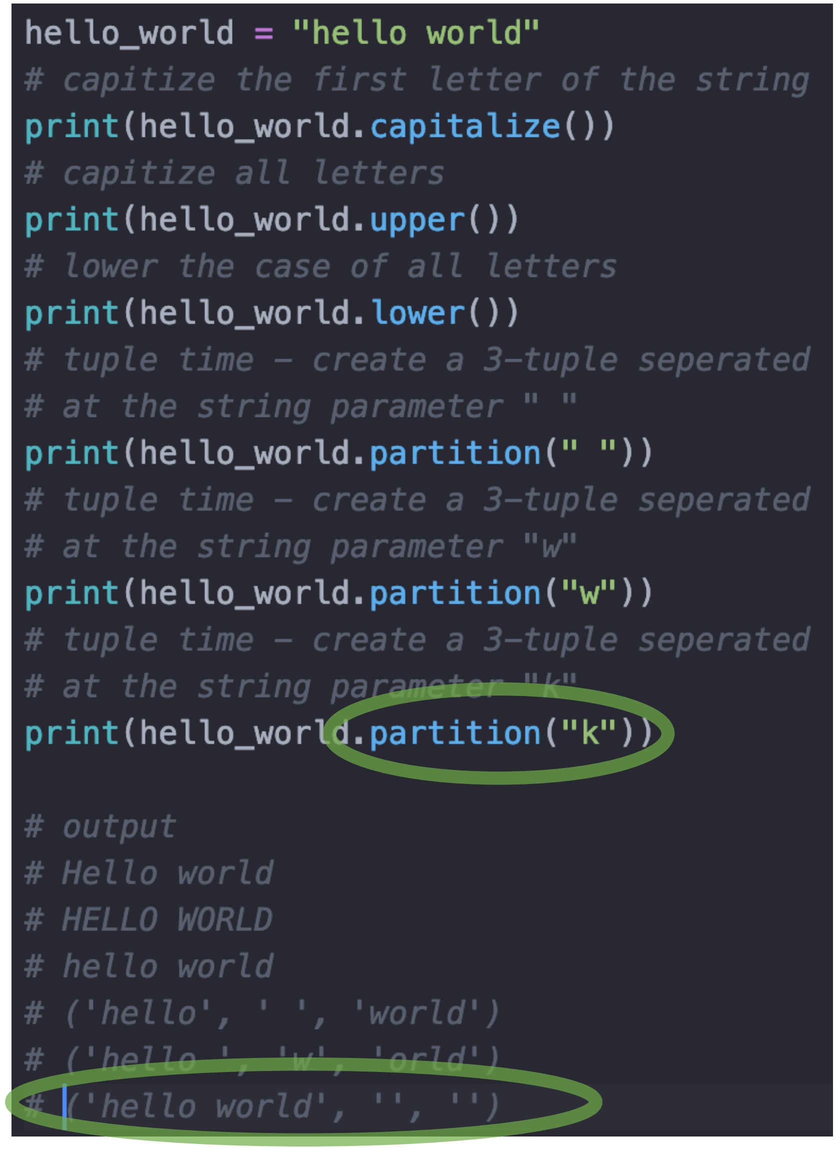
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Functions



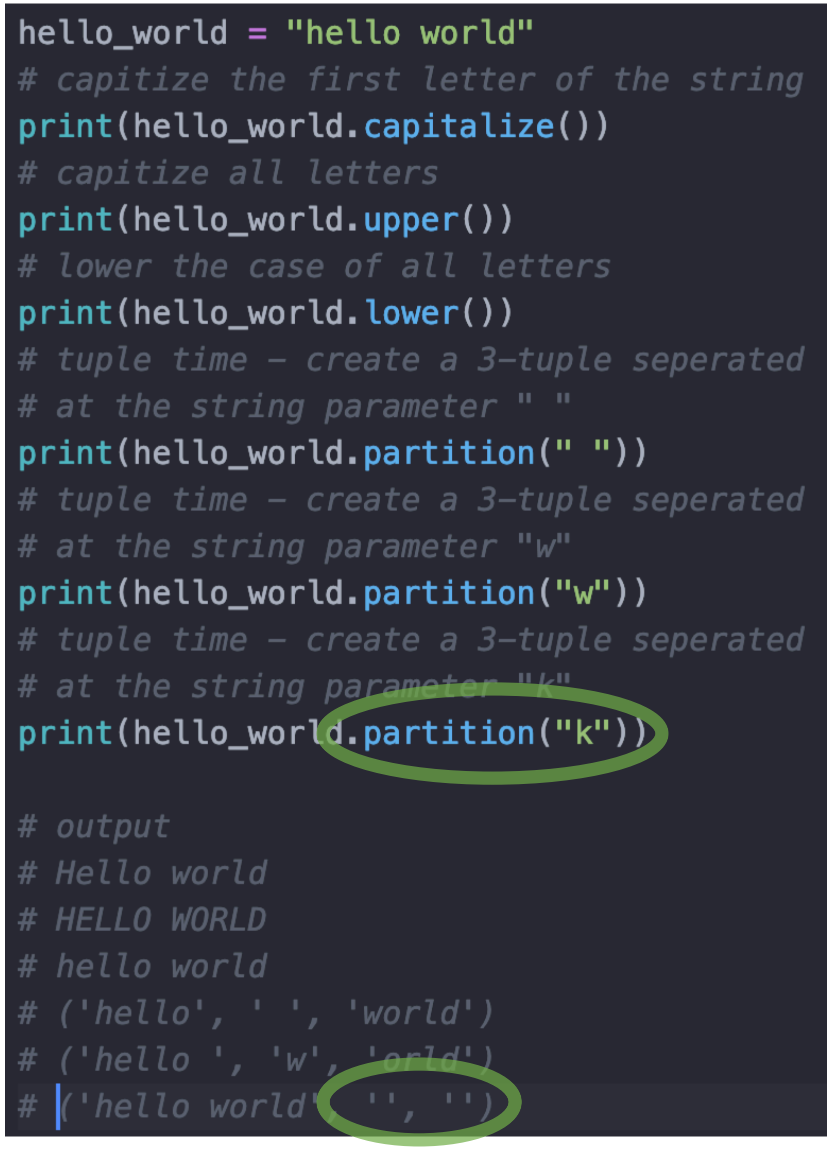
```python
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

String Functions



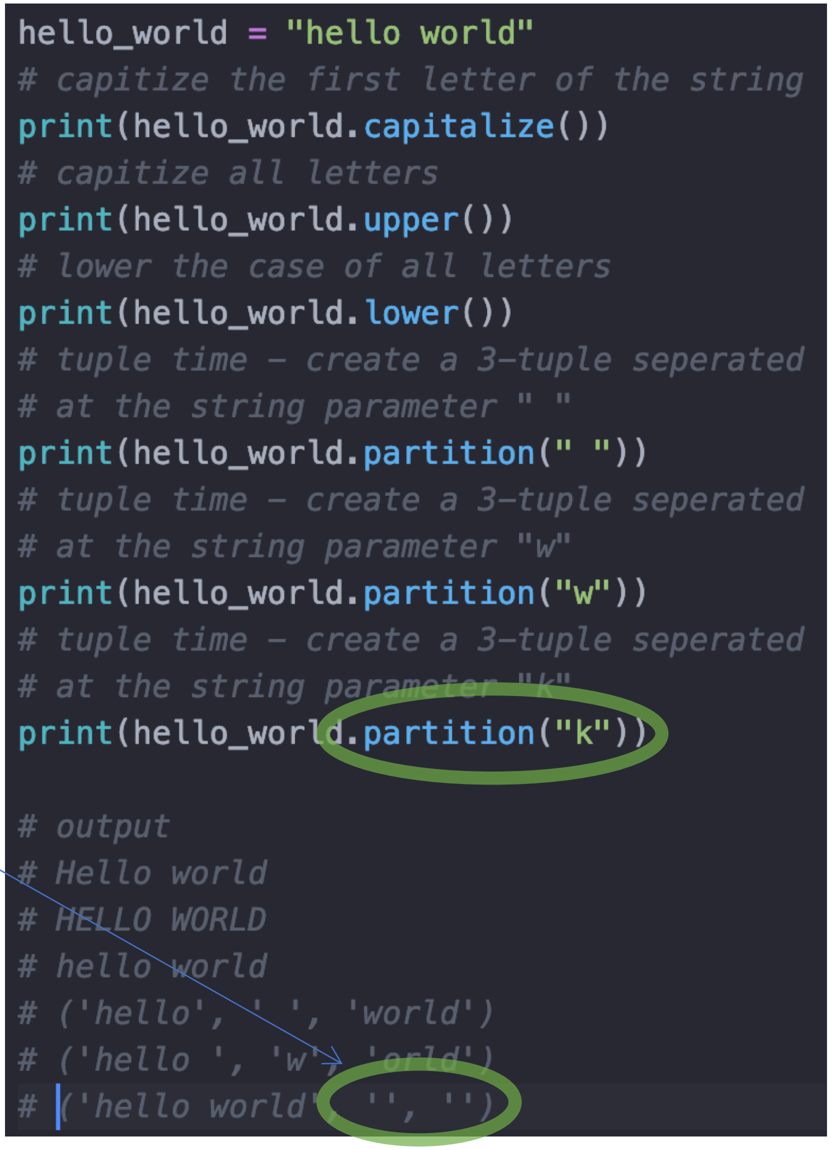
```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

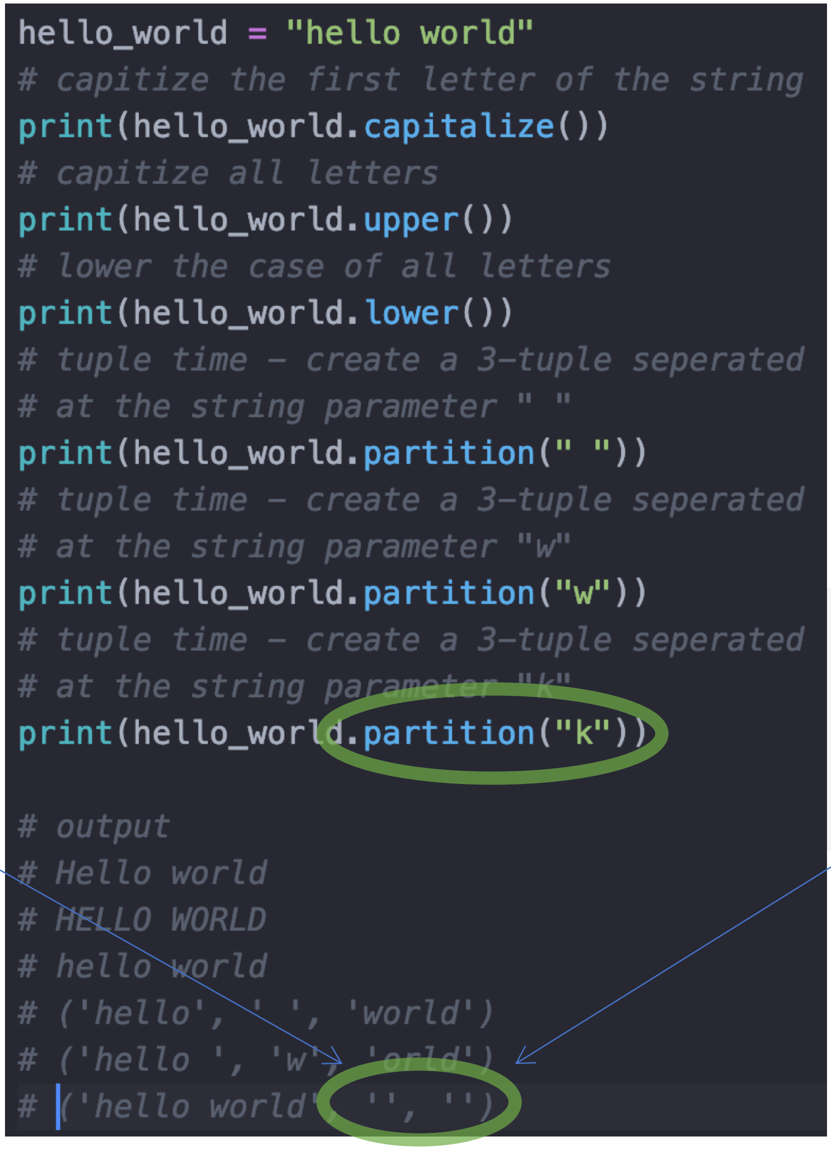
String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

note – this is not a double quoted comma

String Functions

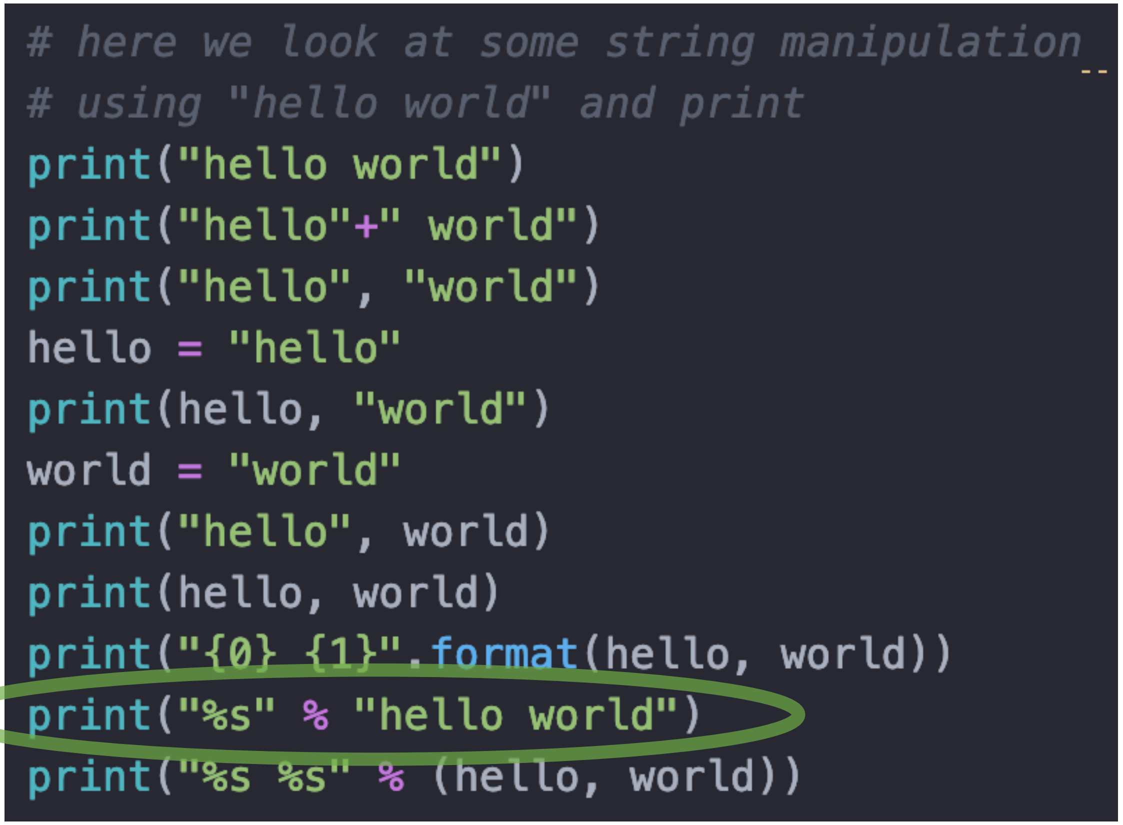


```python
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

note – this is not a double quoted comma

but – two empty single quotes separated by a comma

String Manipulation

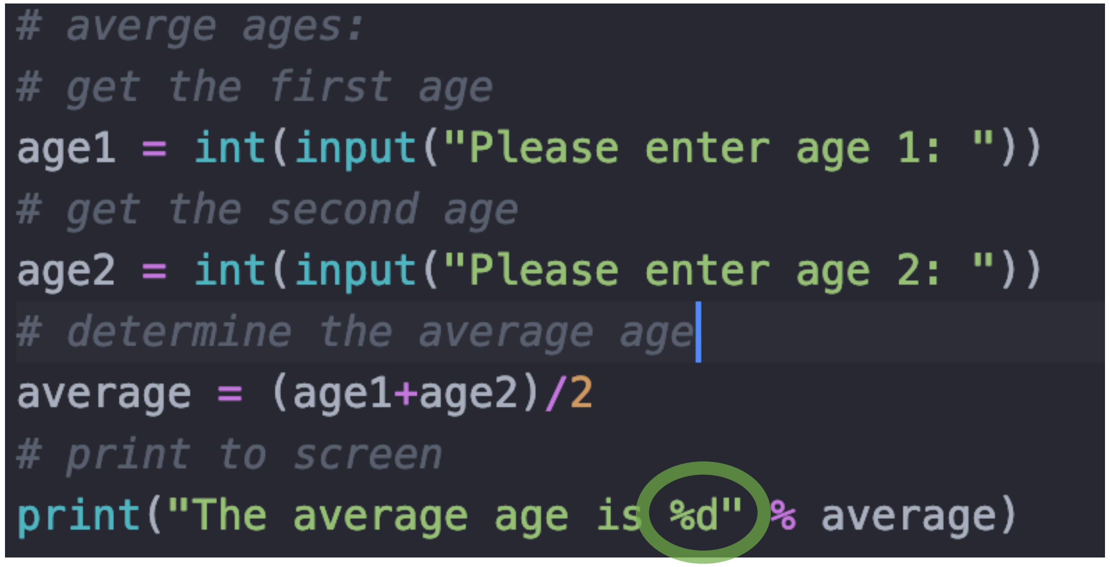


```
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print("hello world") – format the string "hello world" using on the %s operator

Python Functions

We have seen this before:



```
# averge ages:
# get the first age
age1 = int(input("Please enter age 1: "))
# get the second age
age2 = int(input("Please enter age 2: "))
# determine the average age
average = (age1+age2)/2
# print to screen
print("The average age is %d" % average)
```

Where we passed an integer to the string prior to printing it

Python string has a number of operators we can use

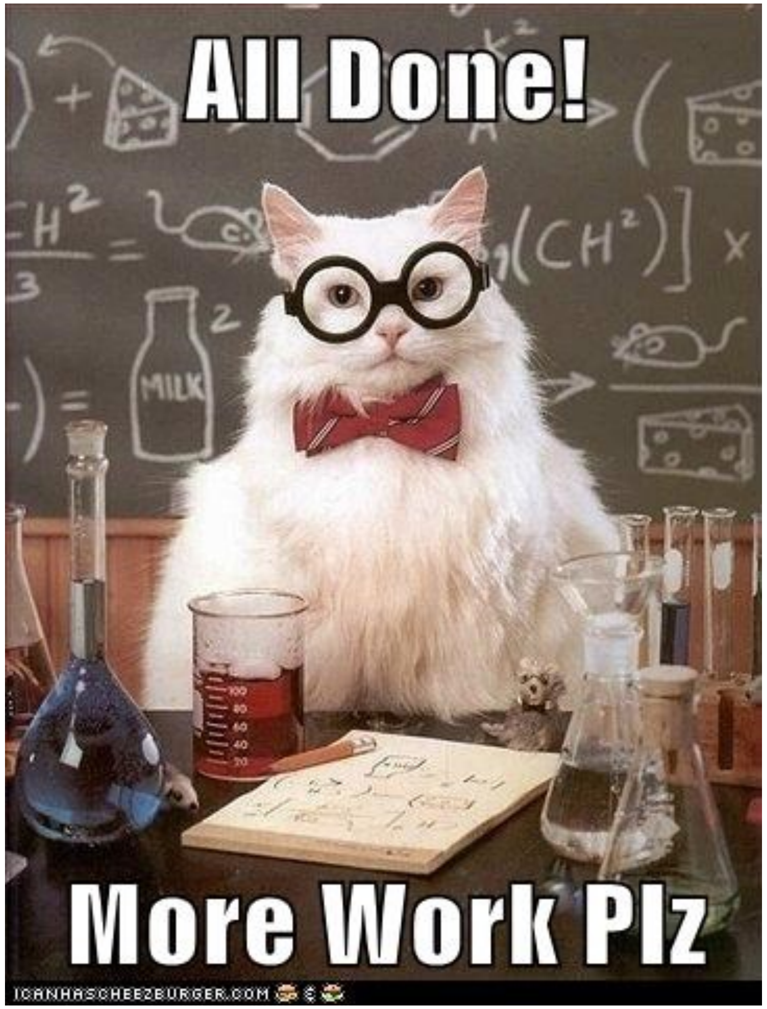
Python Functions

Some of the operators are:

Conversion	Meaning
'd'	Signed integer decimal.
'f'	Floating point decimal format.
'c'	Single character (accepts integer or single character string).
'r'	String (converts any Python object using repr()).
's'	String (converts any Python object using str()).
'a'	String (converts any Python object using ascii()).
'%'	No argument is converted, results in a '%' character in the result.

Canvas Student App

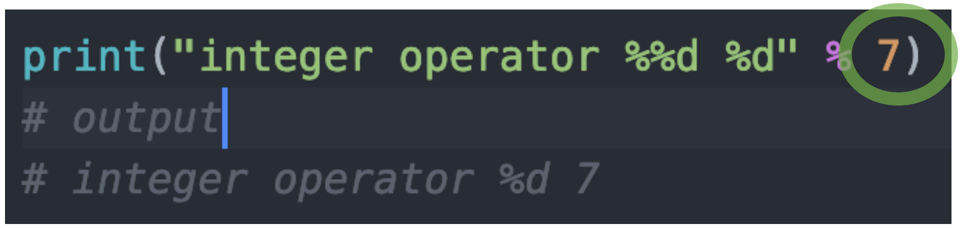
Let’s Sign into this lecture now



University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Functions

Examples:

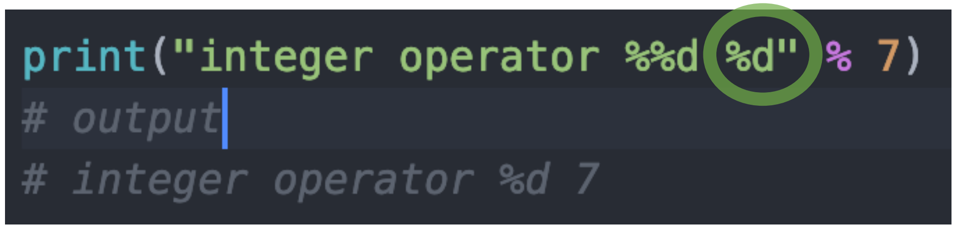


```
print("integer operator %%d %d" % 7)
# output
# integer operator %d 7
```

Here, we print the integer number 7

Python Functions

Examples:



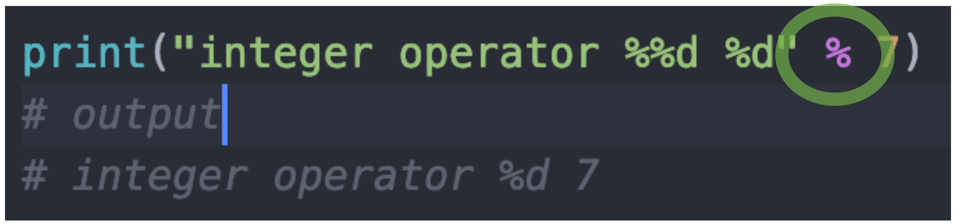
```
print("integer operator %%d %d" % 7)
# output
# integer operator %d 7
```

Here, we print the integer number 7

We use the integer operators %d

Python Functions

Examples:



```
print("integer operator %%d %d" % 7)
# output
# integer operator %d 7
```

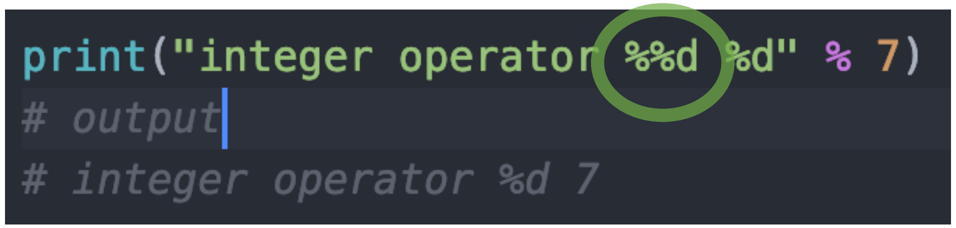
Here, we print the integer number 7

We use the integer operators %d

We use the % operator after the string to tell print() we are using string formatting

Python Functions

Examples:



```
print("integer operator %%d %d" % 7)
# output
# integer operator %d 7
```

Here, we print the integer number 7

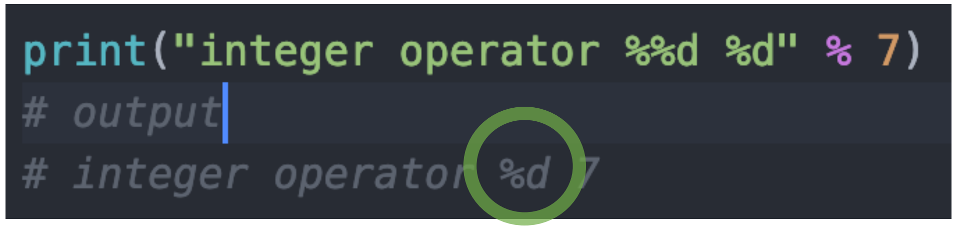
We use the integer operators %d

We use the % operator after the string to tell print() we are using string formatting

We use %% to print a single % when we use string formatting

Python Functions

Examples:



```
print("integer operator %%d %d" % 7)
# output
# integer operator %d 7
```

Here, we print the integer number 7

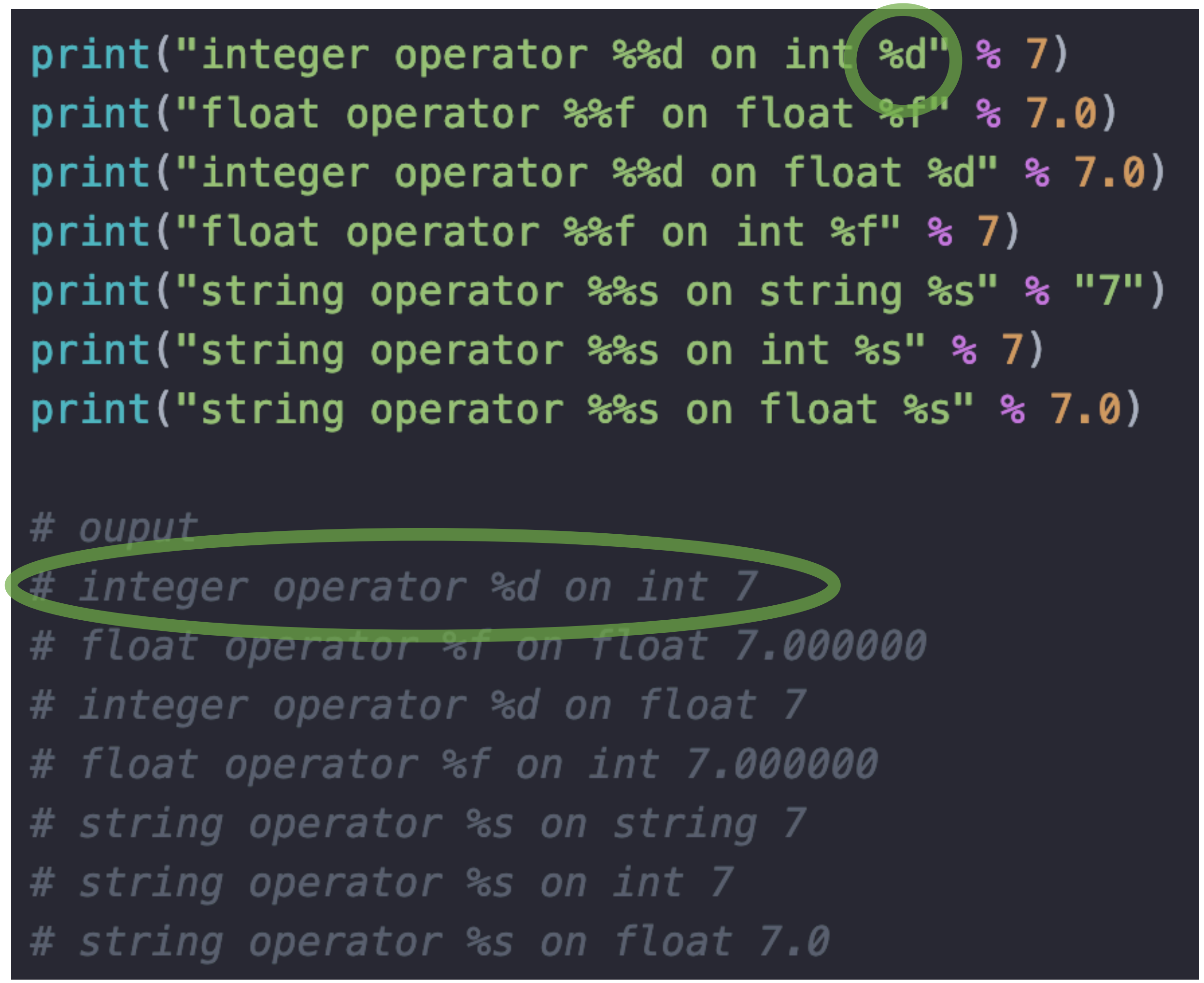
We use the integer operators %d

We use the % operator after the string to tell print() we are using string formatting

We use %% to print a single % when we use string formatting

Python Functions

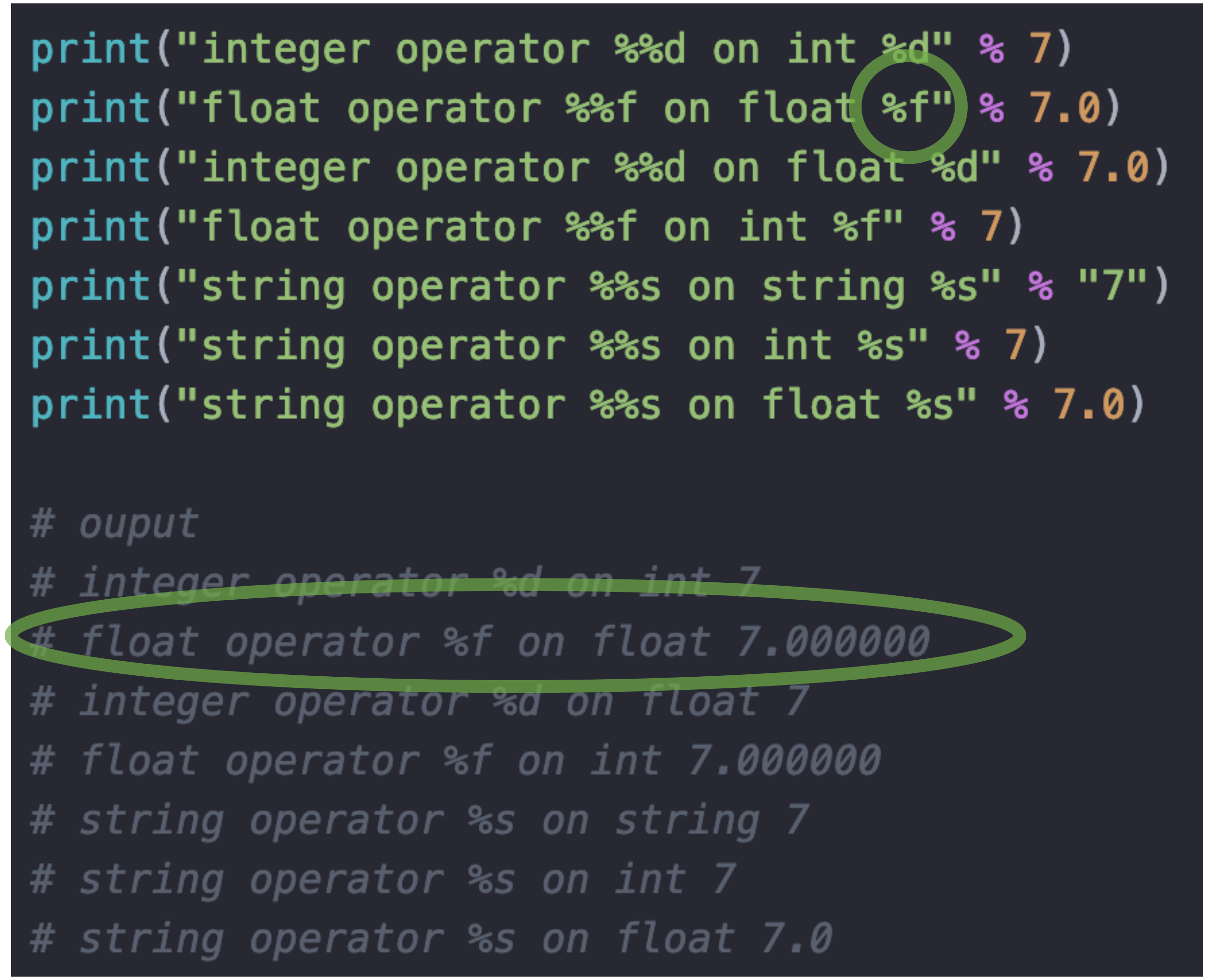
Examples:



```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

Python Functions

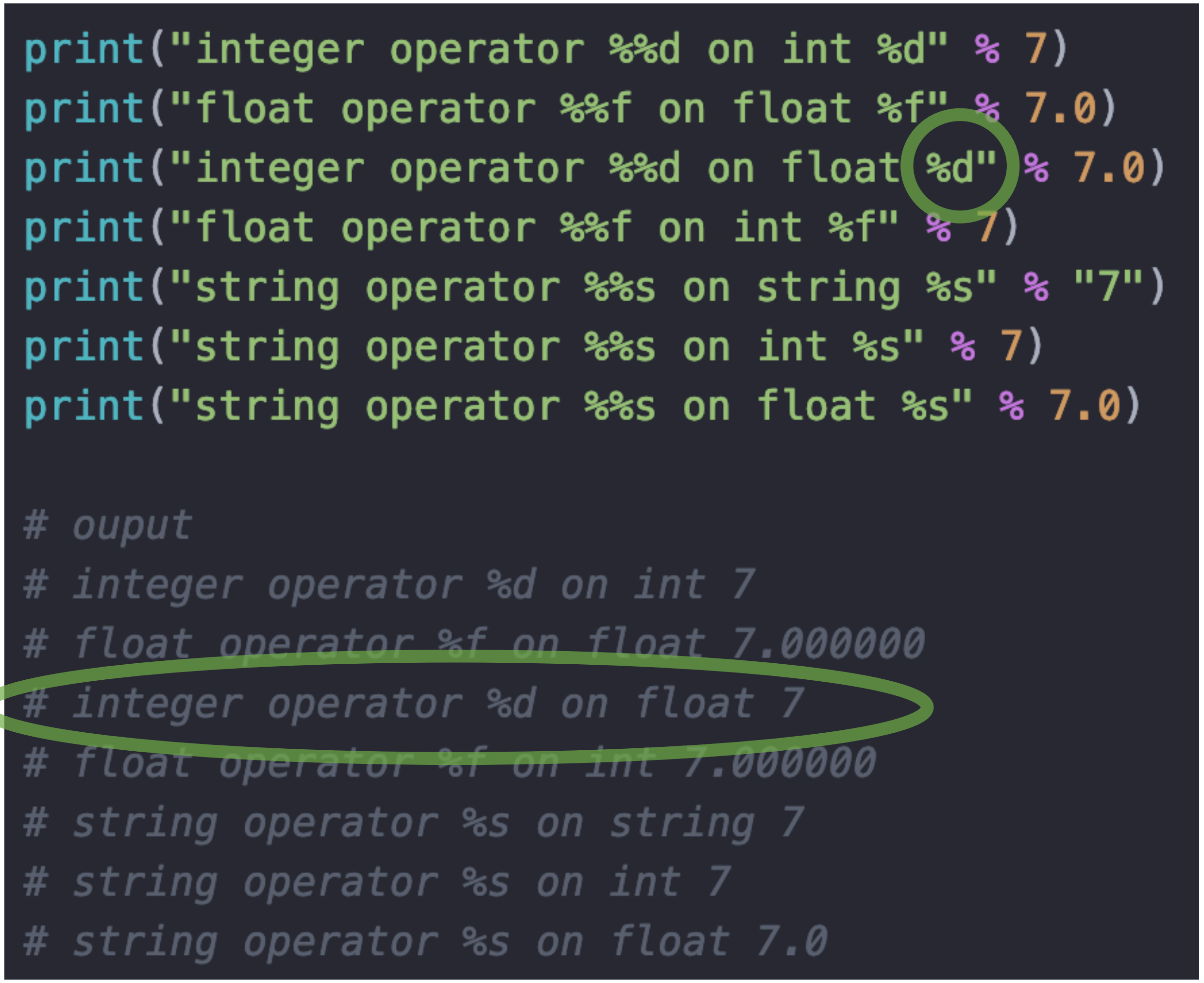
Examples:



```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

Python Functions

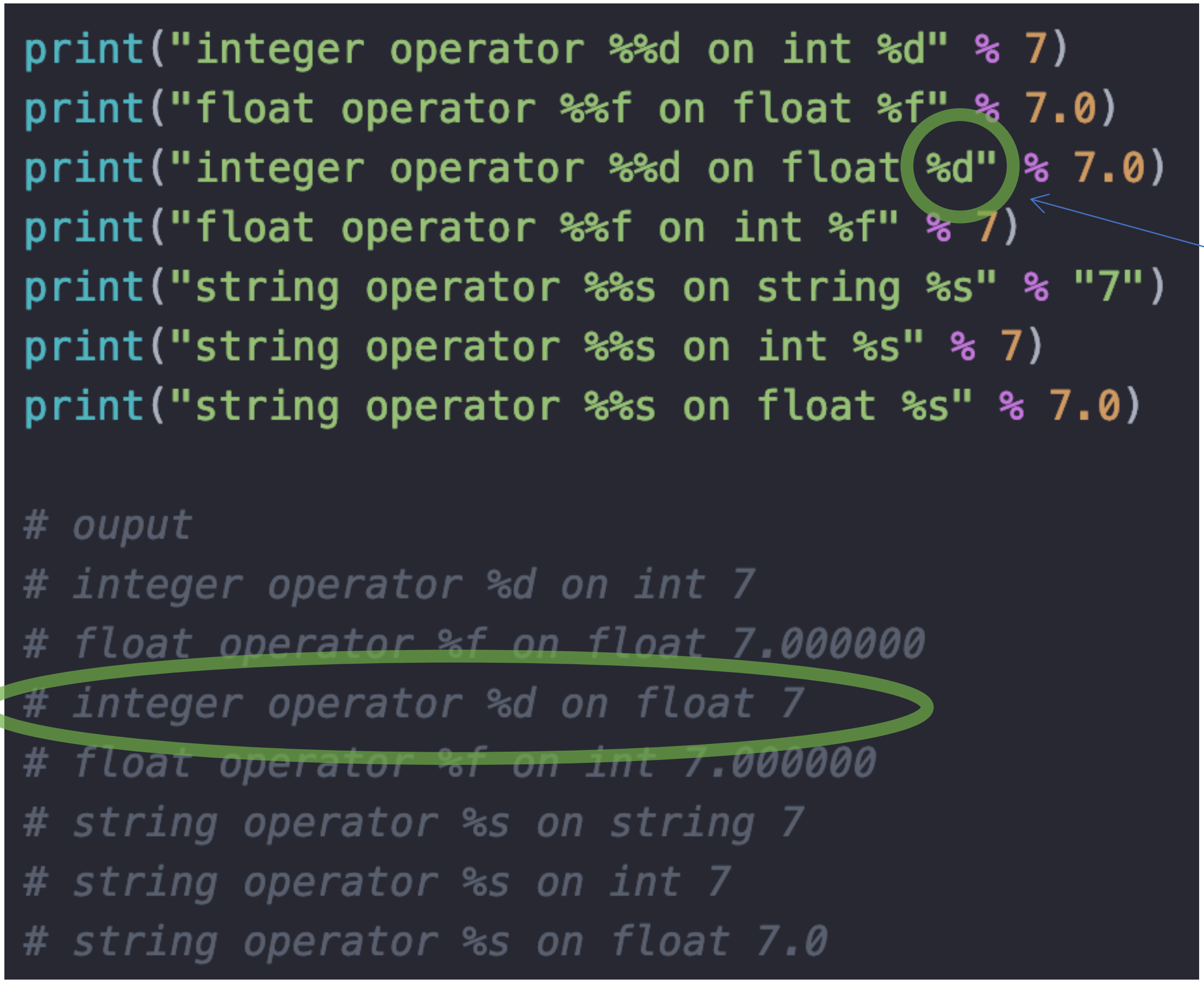
Examples:



```python
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

Python Functions

Examples:

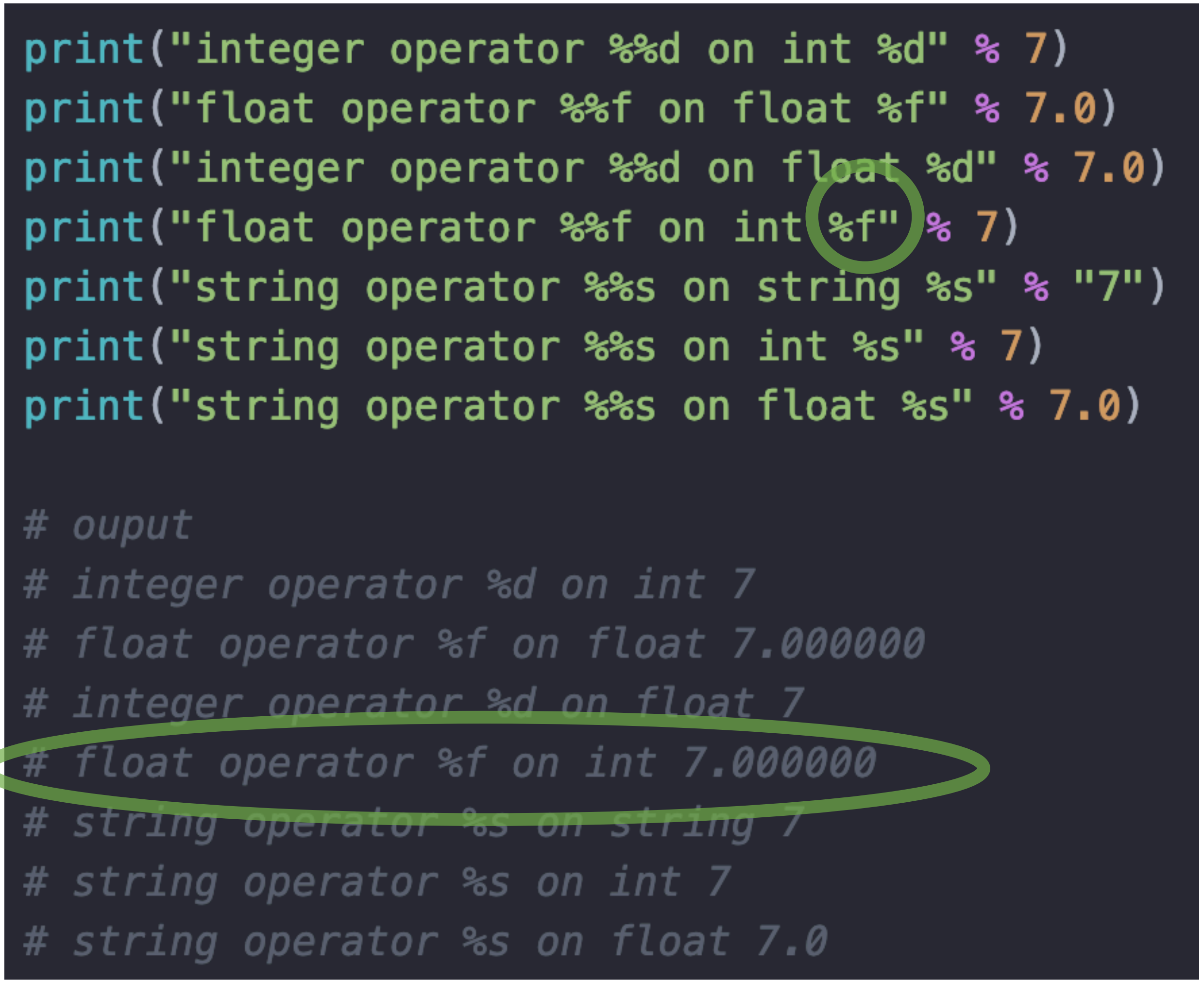


```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

note – this is casting and is the same as using int(7.0)

Python Functions

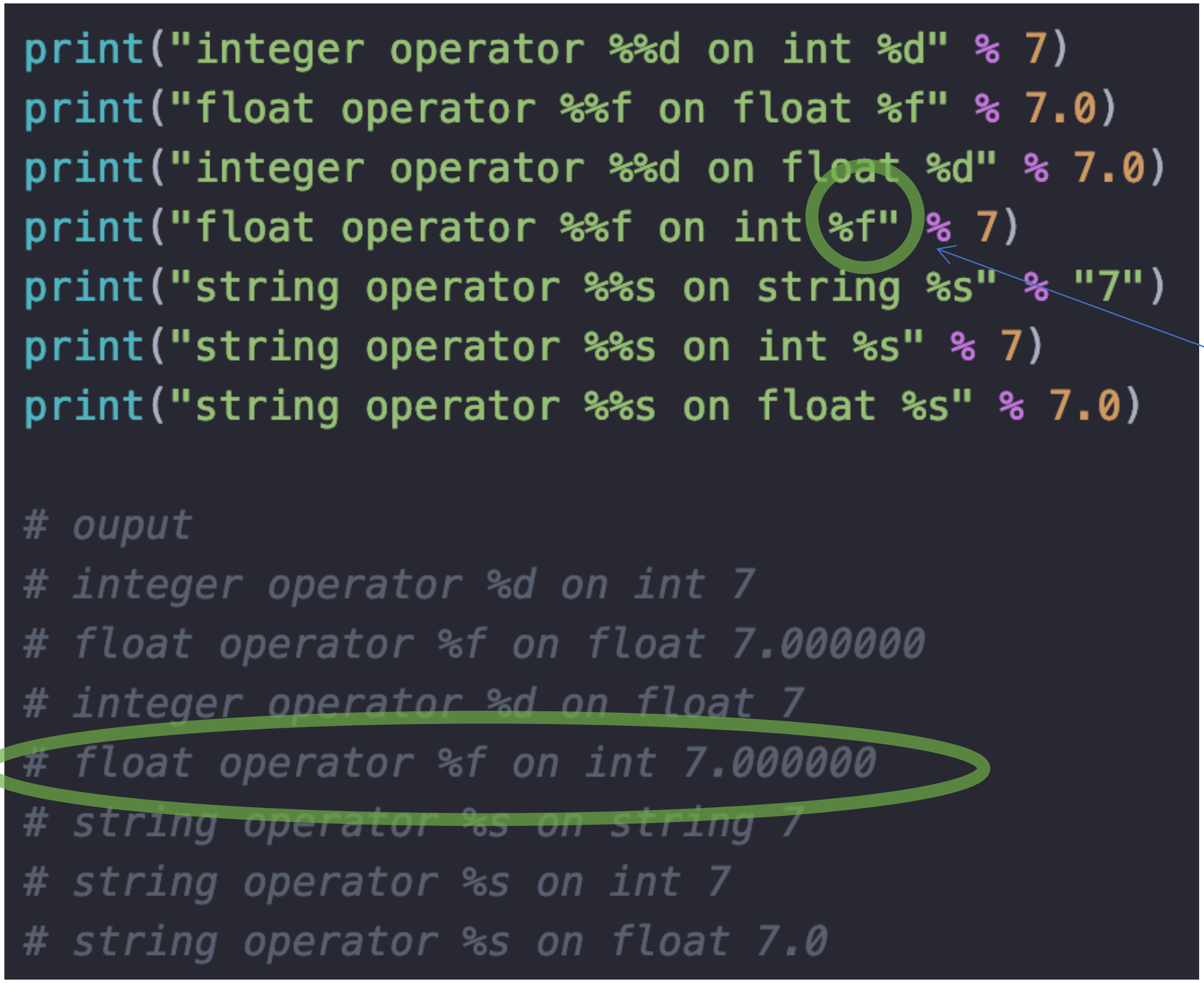
Examples:



```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

Python Functions

Examples:

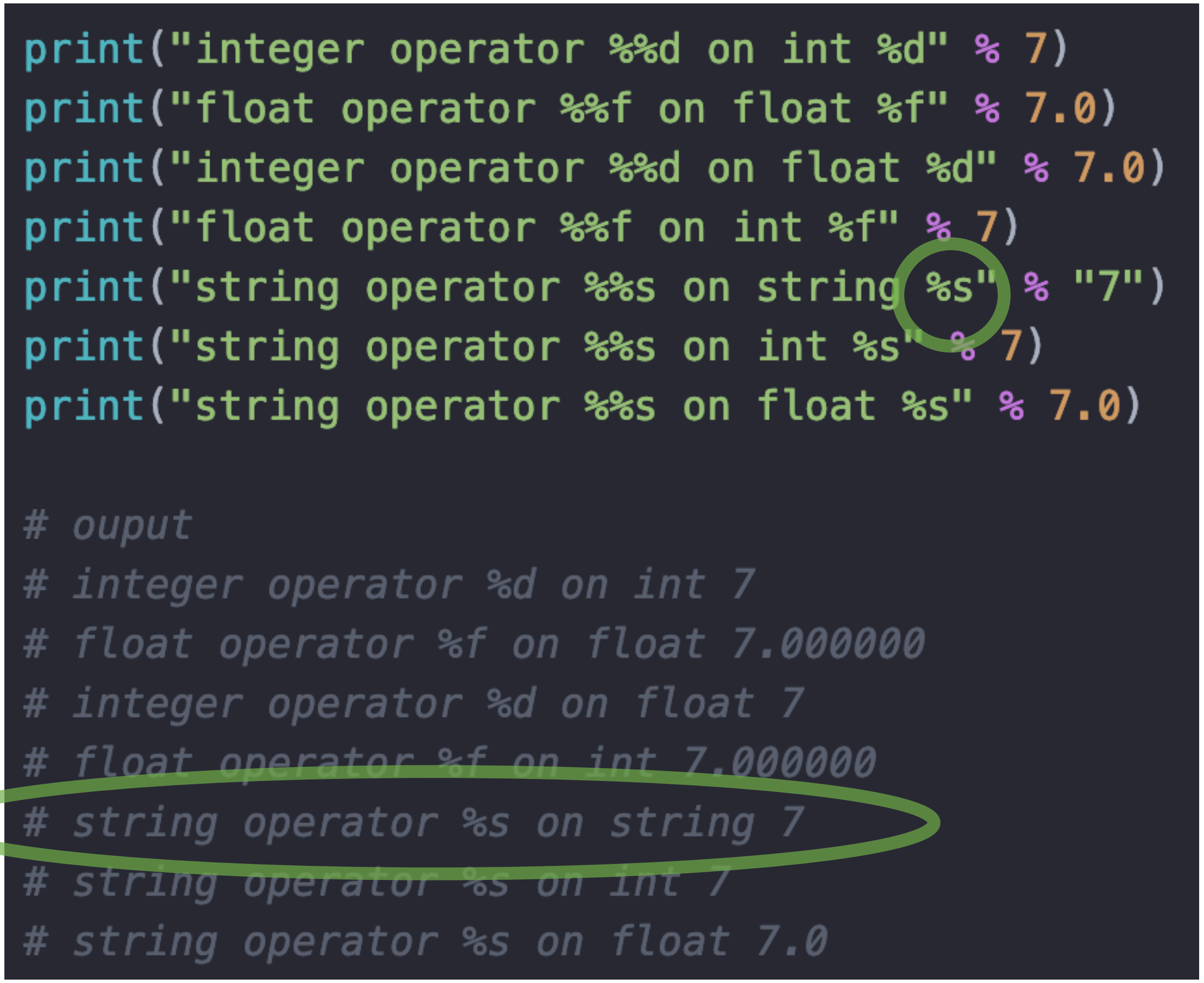


```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

note – this is casting and is the same as using float(7)

Python Functions

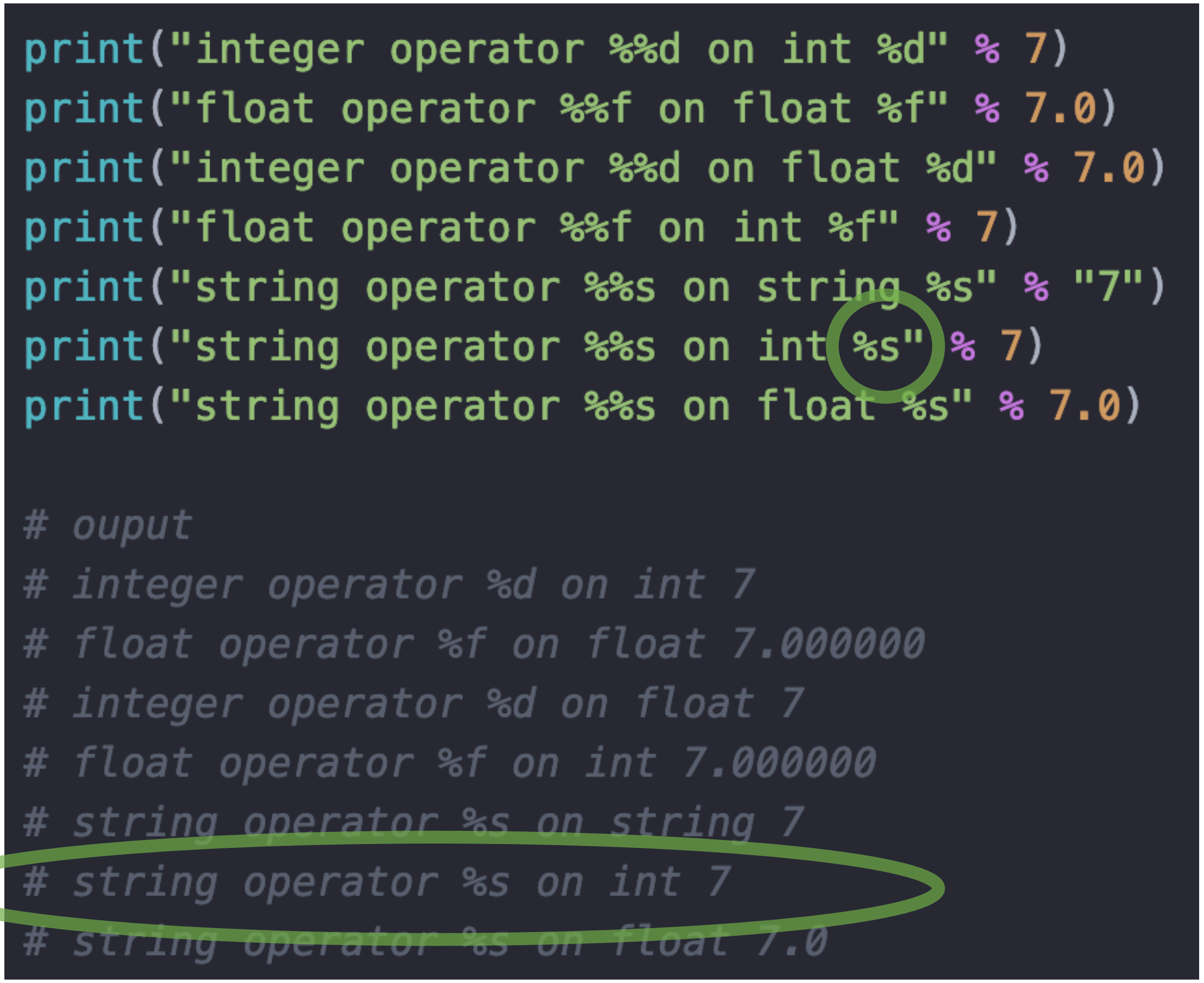
Examples:



```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

Python Functions

Examples:

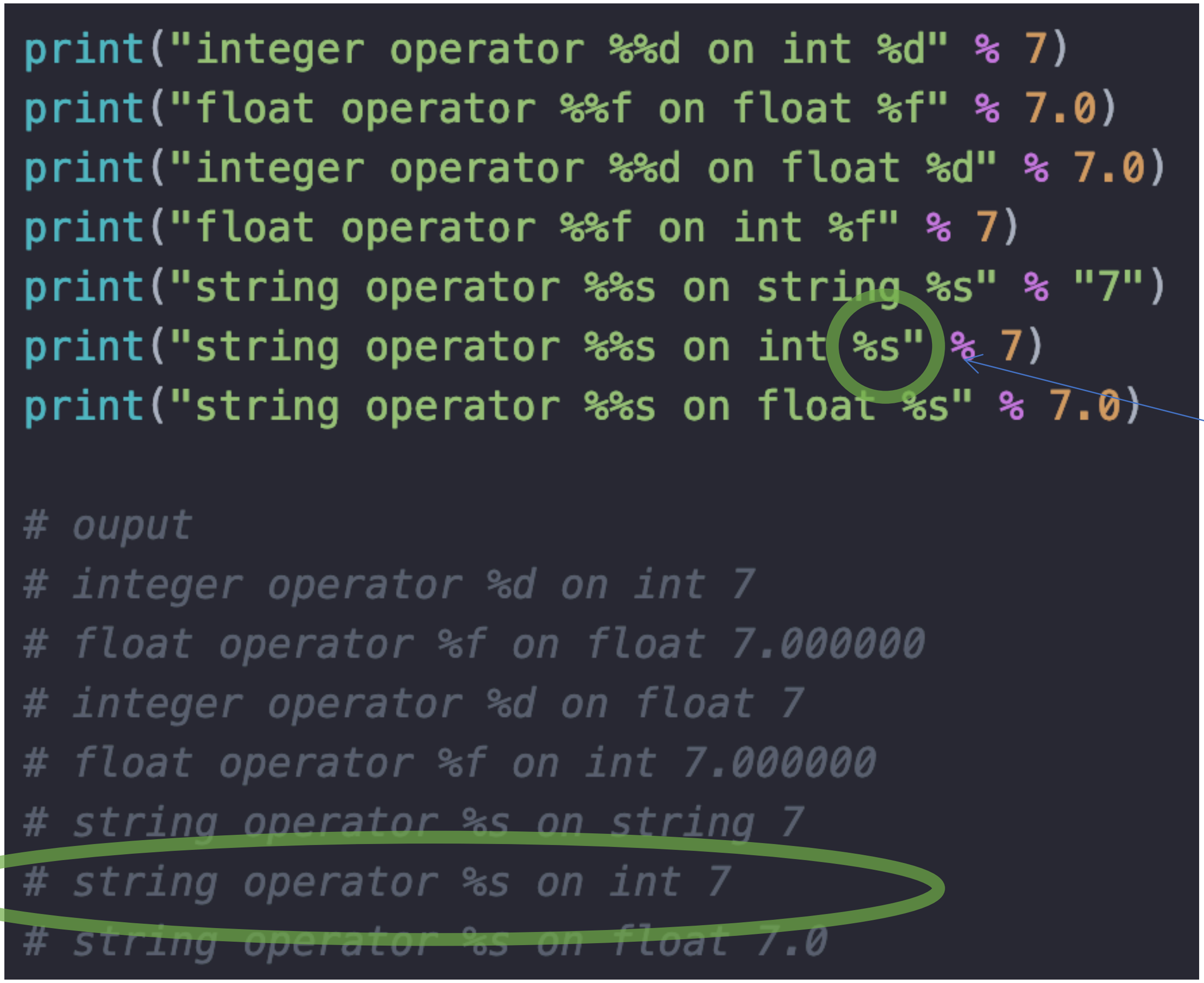


```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Functions

Examples:

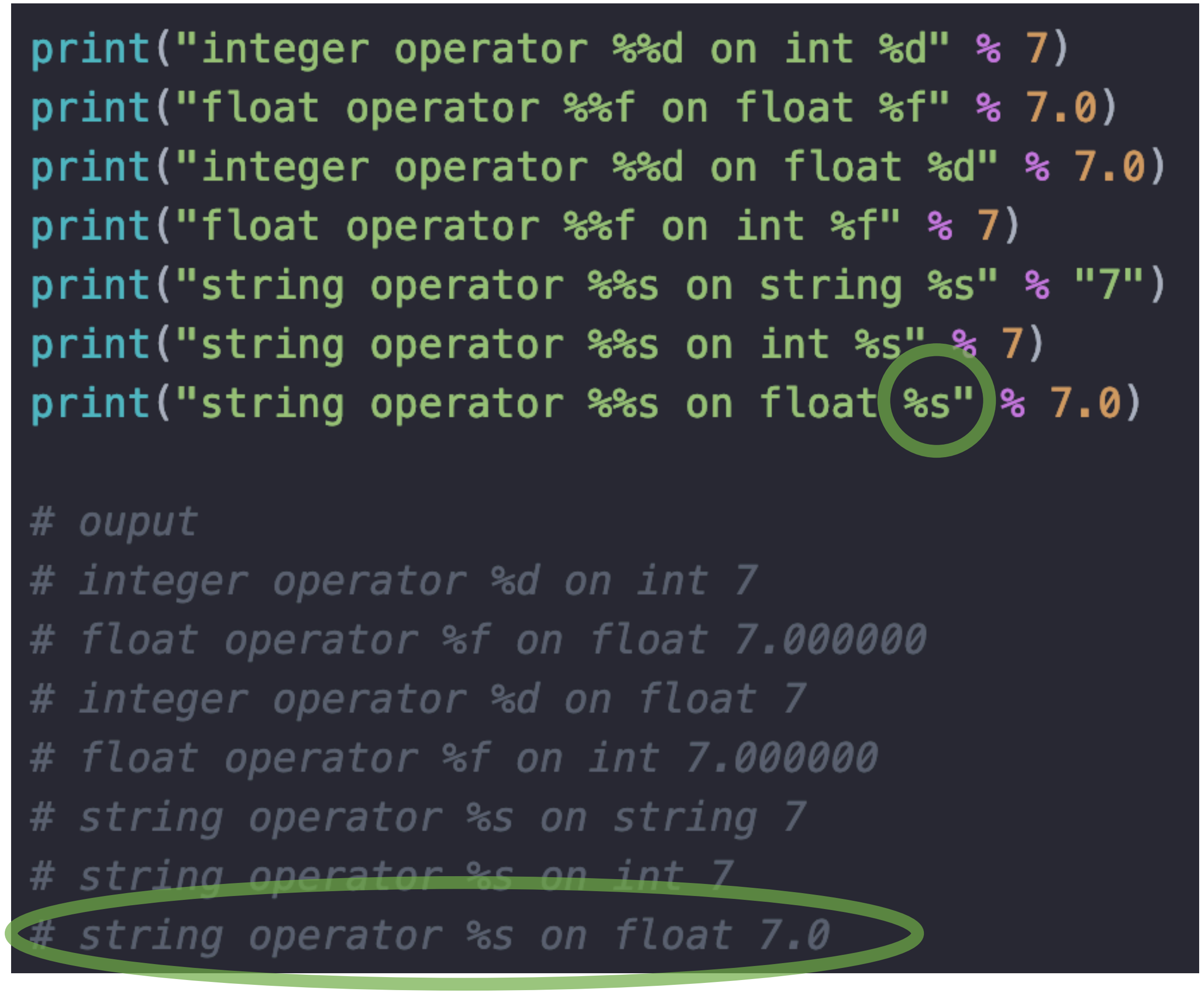


```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

note – this is casting and is the same as using str(7)

Python Functions

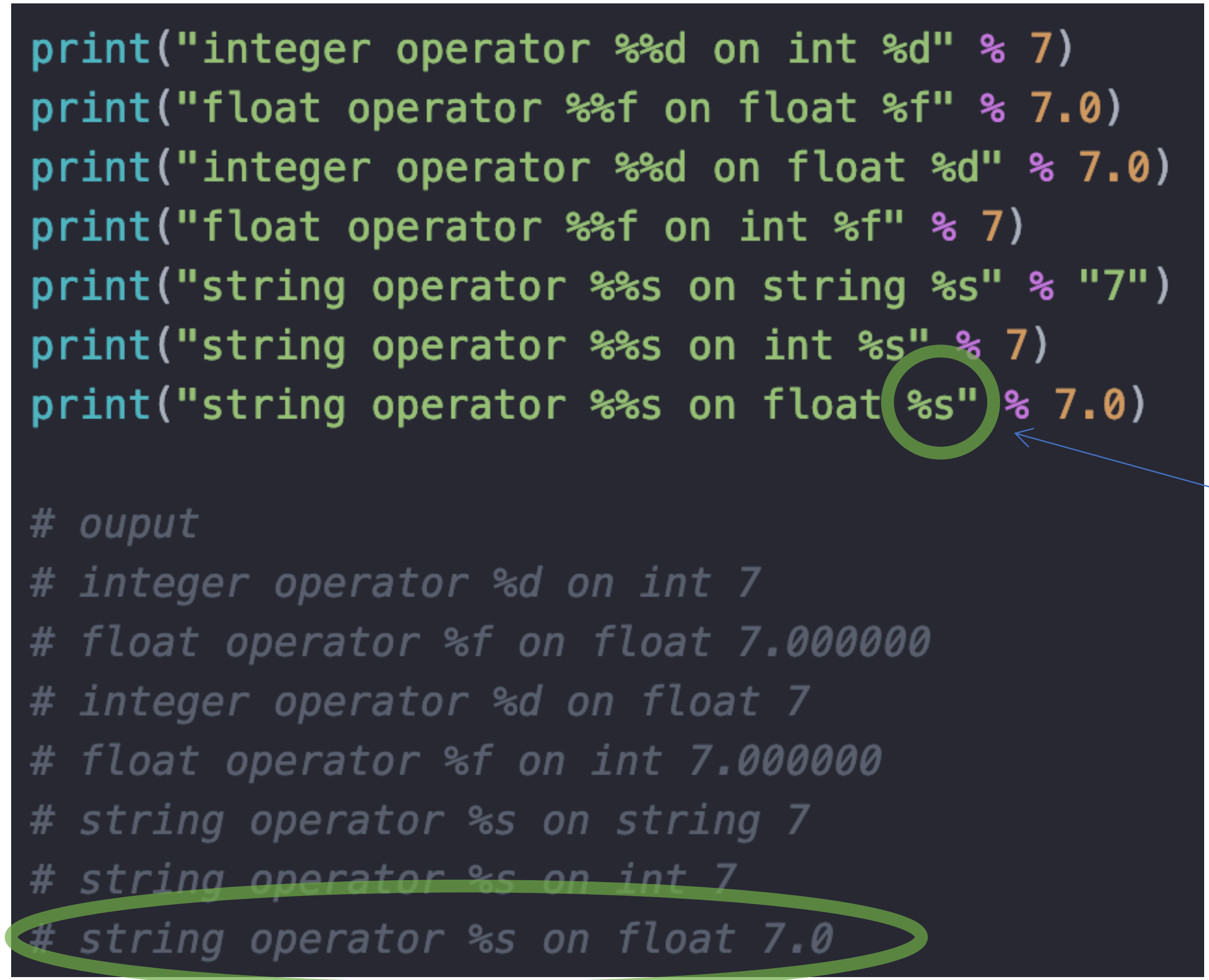
Examples:



```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

Python Functions

Examples:

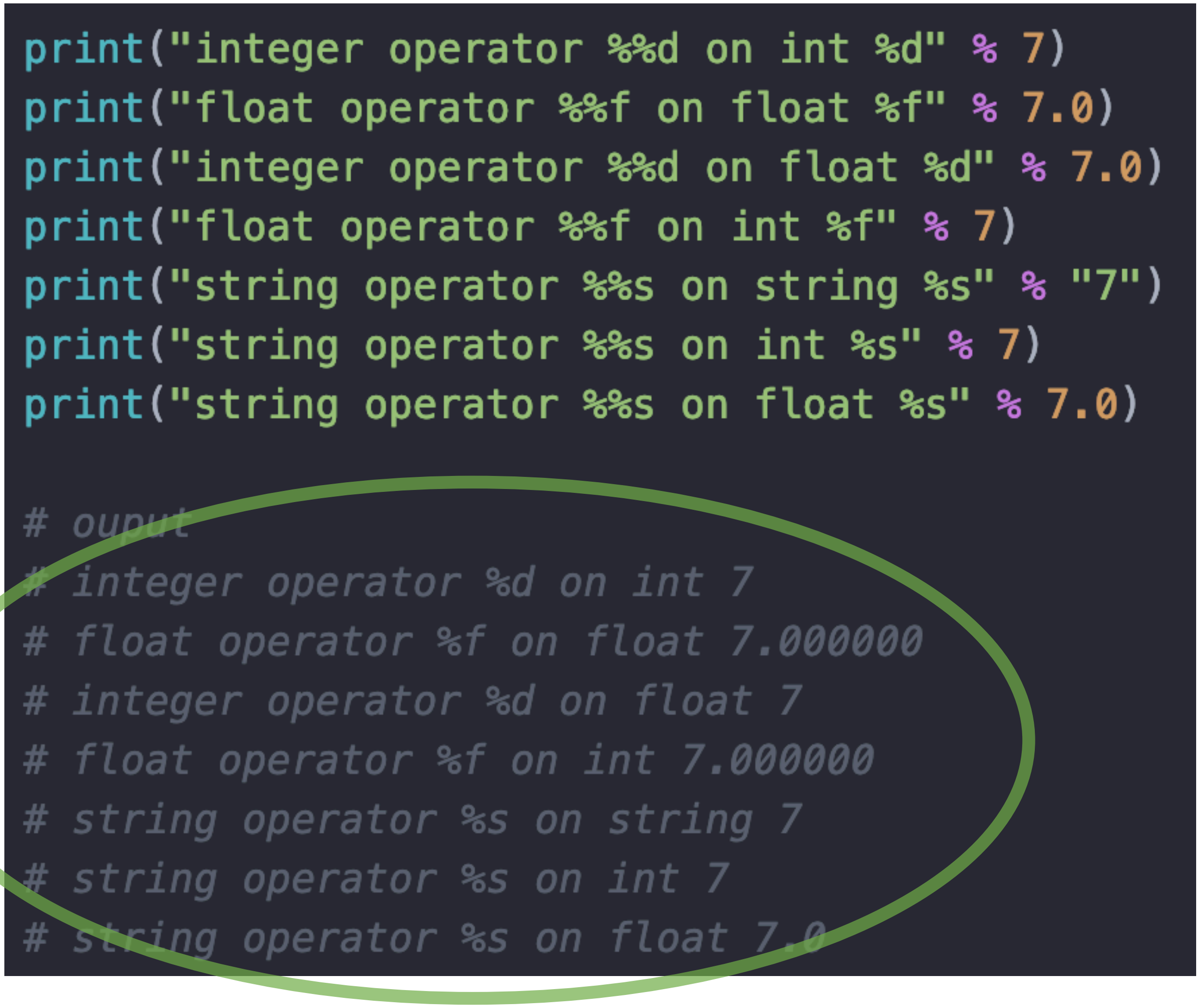


```
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

note – this is casting and is the same as using str(7.0)

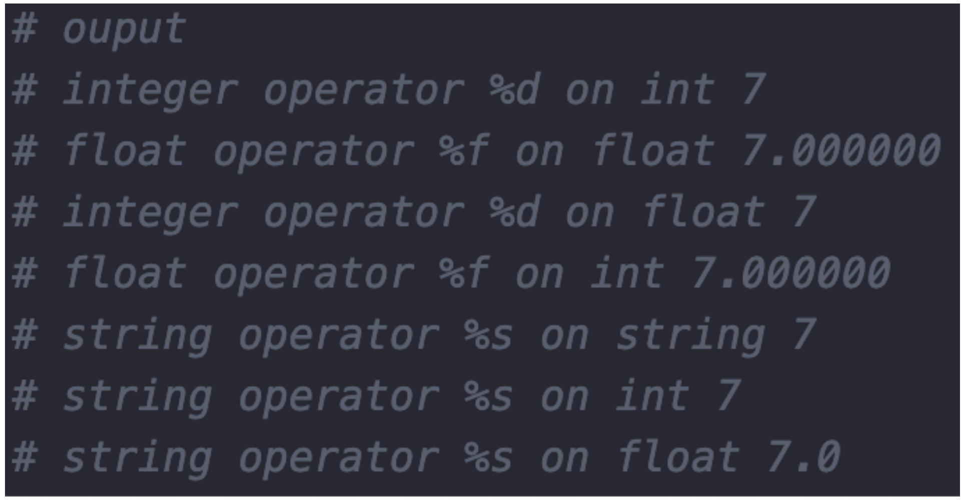
String Formatting

Terrible formatting:



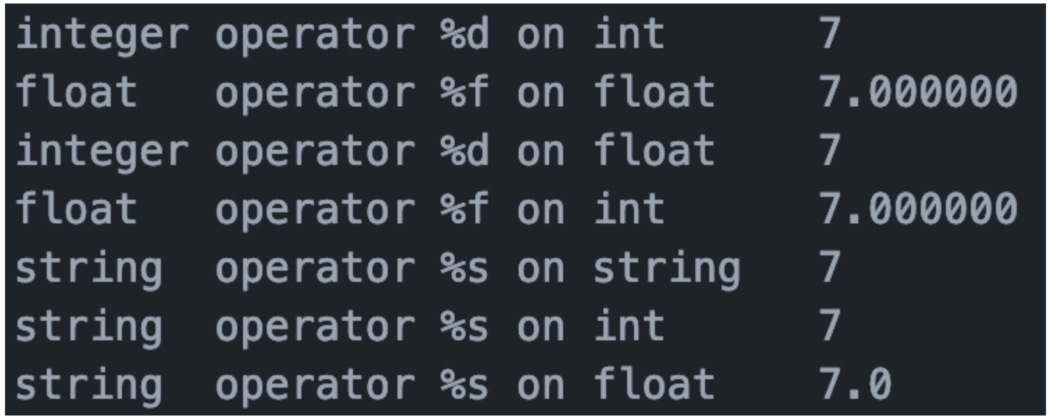
```python
print("integer operator %%d on int %d" % 7)
print("float operator %%f on float %f" % 7.0)
print("integer operator %%d on float %d" % 7.0)
print("float operator %%f on int %f" % 7)
print("string operator %%s on string %s" % "7")
print("string operator %%s on int %s" % 7)
print("string operator %%s on float %s" % 7.0)

# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

String Formatting



```
# ouput
# integer operator %d on int 7
# float operator %f on float 7.000000
# integer operator %d on float 7
# float operator %f on int 7.000000
# string operator %s on string 7
# string operator %s on int 7
# string operator %s on float 7.0
```

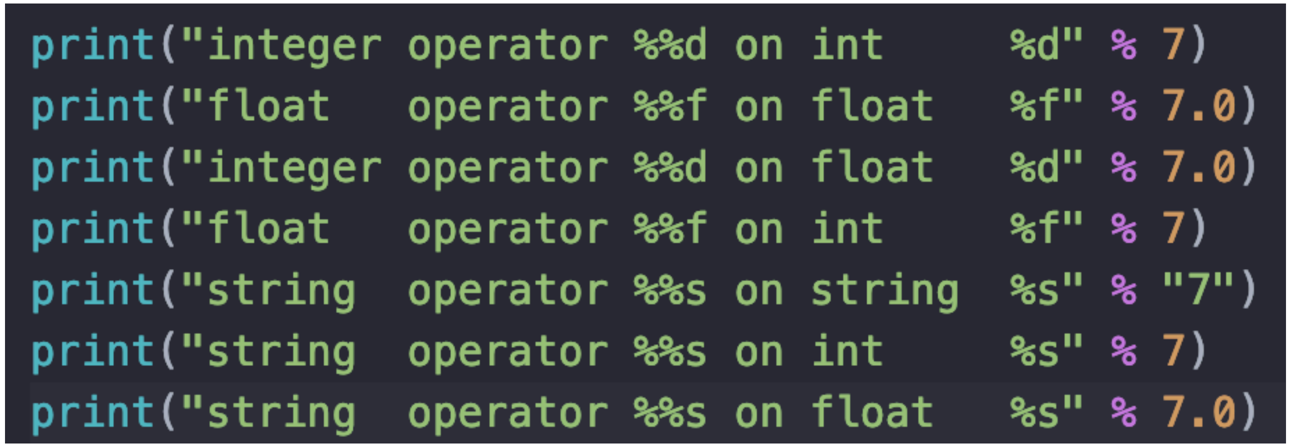
This output would be nice



```
integer operator %d on int       7
float   operator %f on float     7.000000
integer operator %d on float     7
float   operator %f on int       7.000000
string  operator %s on string    7
string  operator %s on int       7
string  operator %s on float     7.0
```

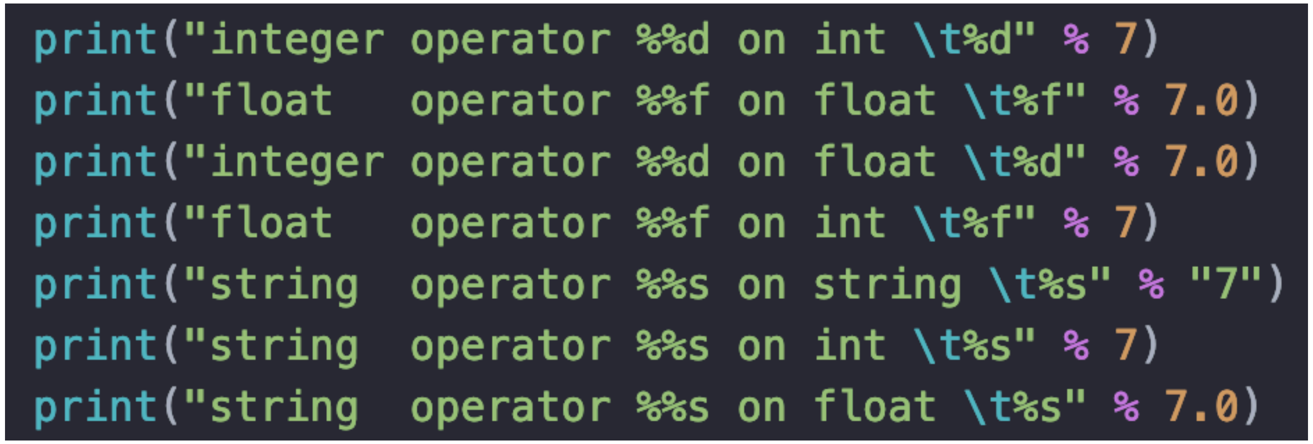
String Formatting

We can either – add spaces:



```
print("integer operator %%d on int     %d" % 7)
print("float   operator %%f on float   %f" % 7.0)
print("integer operator %%d on float   %d" % 7.0)
print("float   operator %%f on int     %f" % 7)
print("string  operator %%s on string  %s" % "7")
print("string  operator %%s on int     %s" % 7)
print("string  operator %%s on float   %s" % 7.0)
```

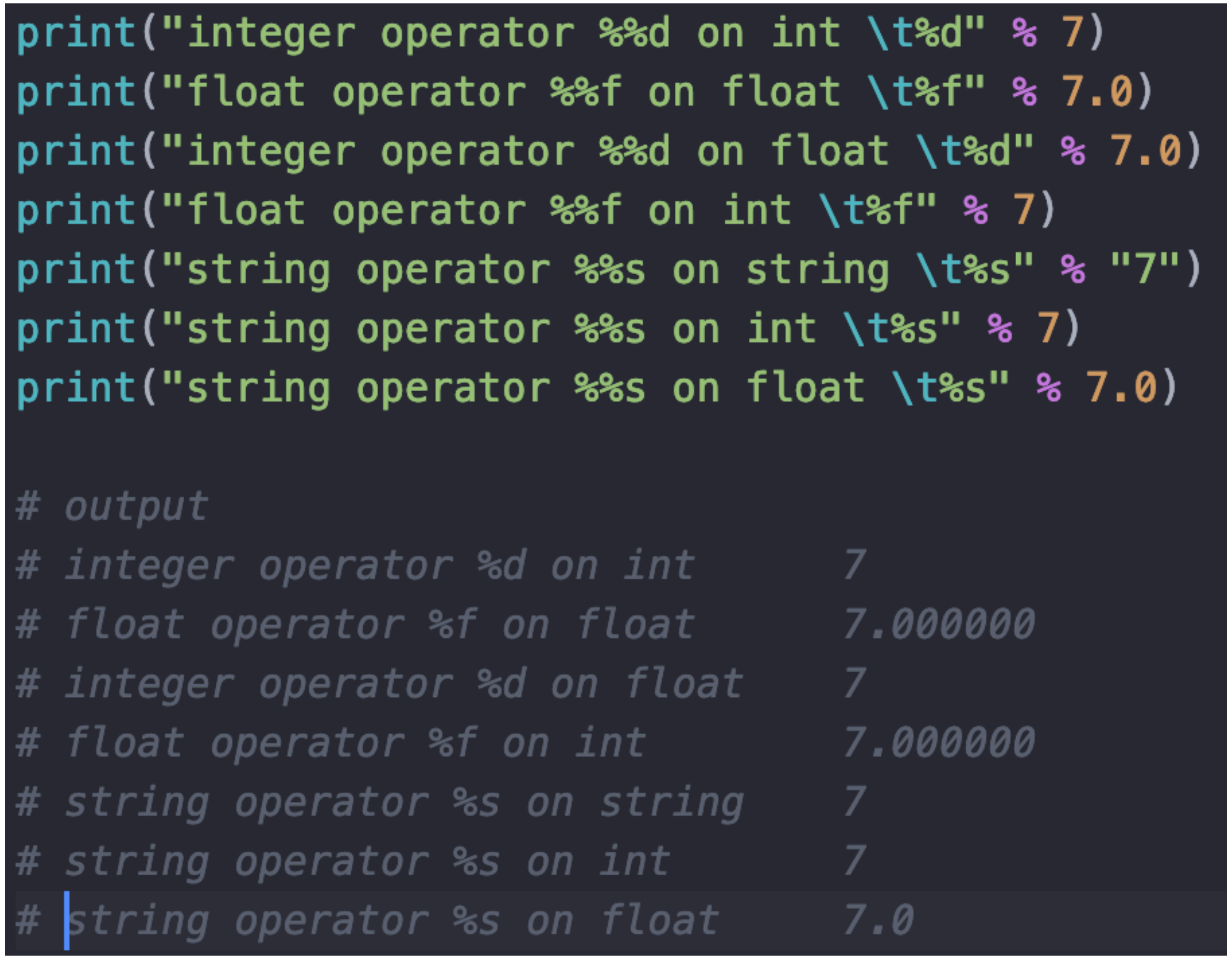
Or we can start to format the output string



```
print("integer operator %%d on int \t%d" % 7)
print("float   operator %%f on float \t%f" % 7.0)
print("integer operator %%d on float \t%d" % 7.0)
print("float   operator %%f on int \t%f" % 7)
print("string  operator %%s on string \t%s" % "7")
print("string  operator %%s on int \t%s" % 7)
print("string  operator %%s on float \t%s" % 7.0)
```

String Formatting

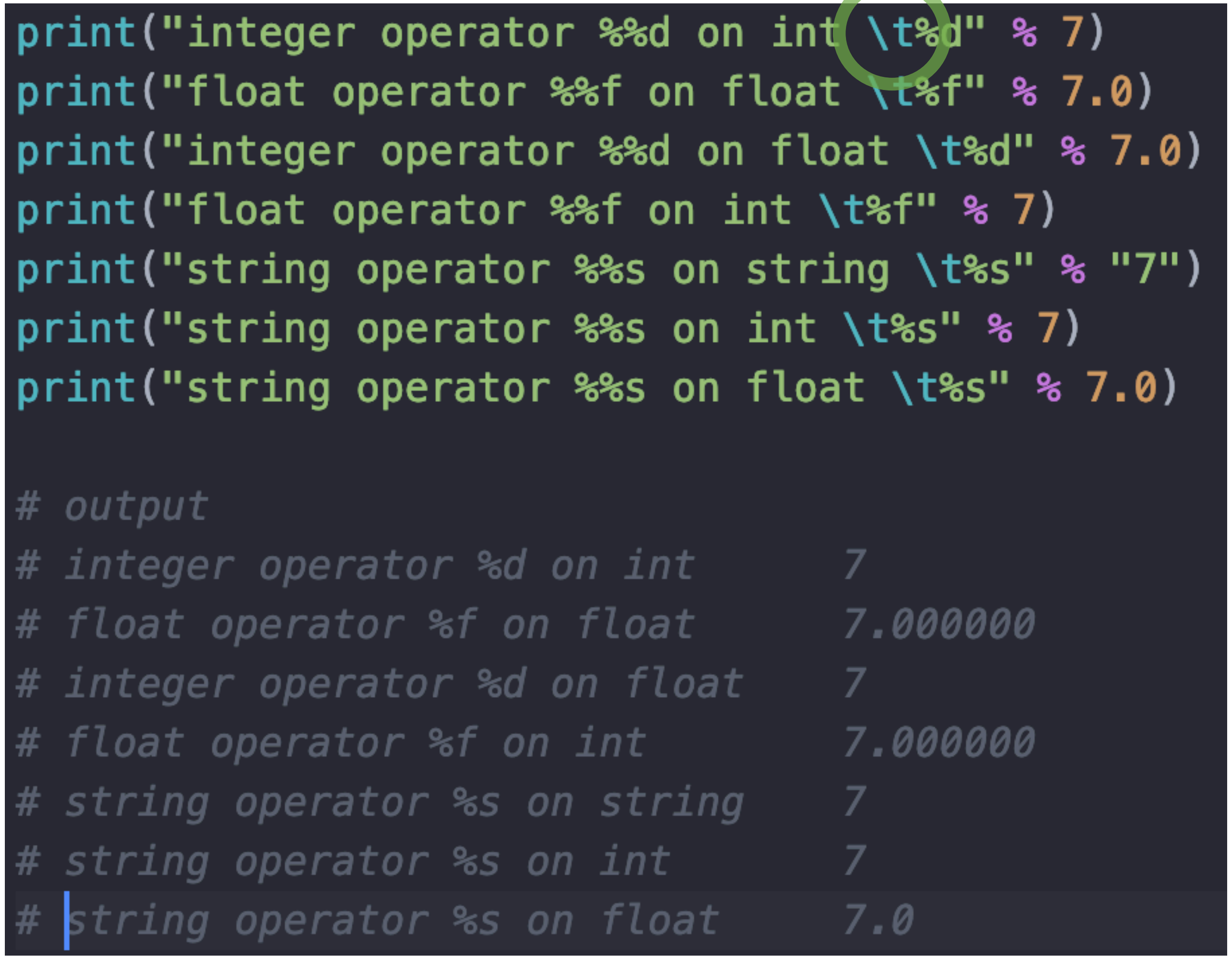
In this example - \t adds a tab to our output string



```
print("integer operator %%d on int \t%d" % 7)
print("float operator %%f on float \t%f" % 7.0)
print("integer operator %%d on float \t%d" % 7.0)
print("float operator %%f on int \t%f" % 7)
print("string operator %%s on string \t%s" % "7")
print("string operator %%s on int \t%s" % 7)
print("string operator %%s on float \t%s" % 7.0)

# output
# integer operator %d on int      7
# float operator %f on float      7.000000
# integer operator %d on float    7
# float operator %f on int        7.000000
# string operator %s on string    7
# string operator %s on int       7
# string operator %s on float     7.0
```

String Formatting

In this example - \t adds a tab to our output string



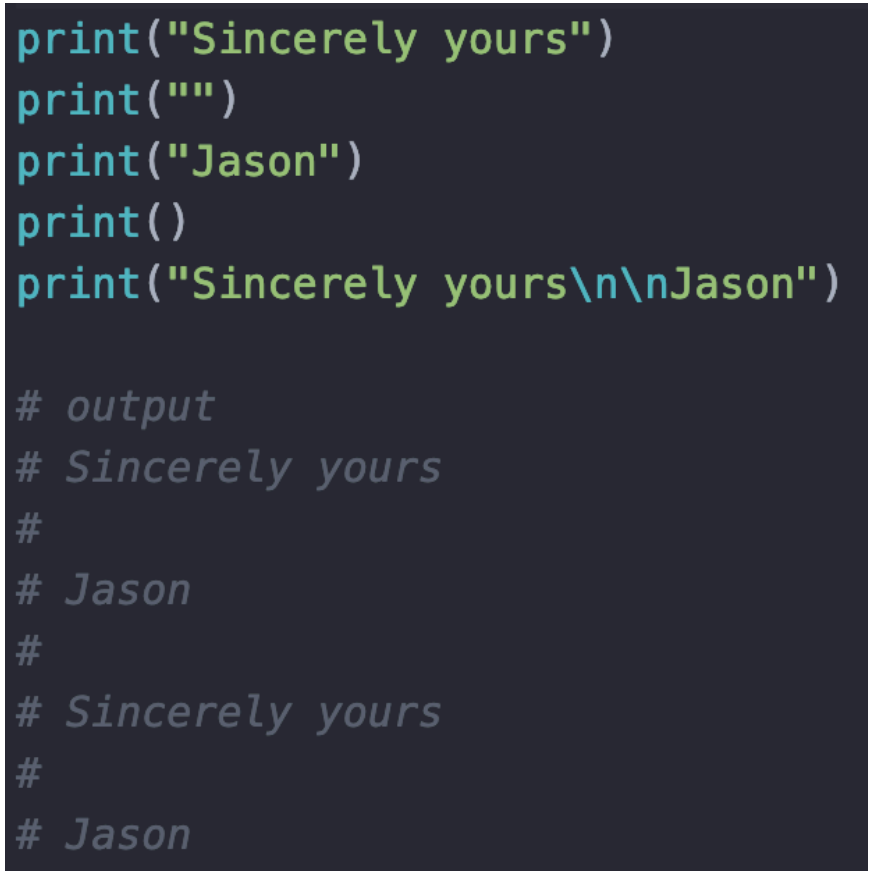
```python
print("integer operator %%d on int \t%d" % 7)
print("float operator %%f on float \t%f" % 7.0)
print("integer operator %%d on float \t%d" % 7.0)
print("float operator %%f on int \t%f" % 7)
print("string operator %%s on string \t%s" % "7")
print("string operator %%s on int \t%s" % 7)
print("string operator %%s on float \t%s" % 7.0)

# output
# integer operator %d on int      7
# float operator %f on float      7.000000
# integer operator %d on float    7
# float operator %f on int        7.000000
# string operator %s on string    7
# string operator %s on int       7
# string operator %s on float     7.0
```

String Formatting

One other commonly used string format is \n (newline)

String Formatting

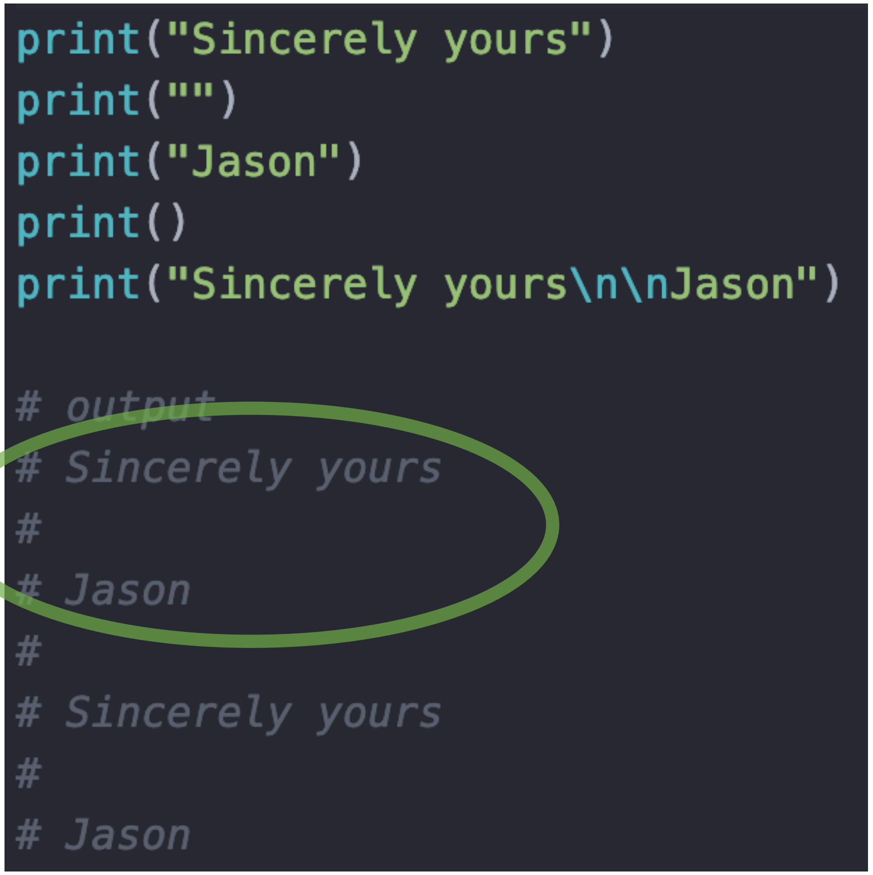
Say I want to sign off at the end of a letter



```python
print("Sincerely yours")
print("")
print("Jason")
print()
print("Sincerely yours\n\nJason")

# output
# Sincerely yours
#
# Jason
#
# Sincerely yours
#
# Jason
```

String Formatting

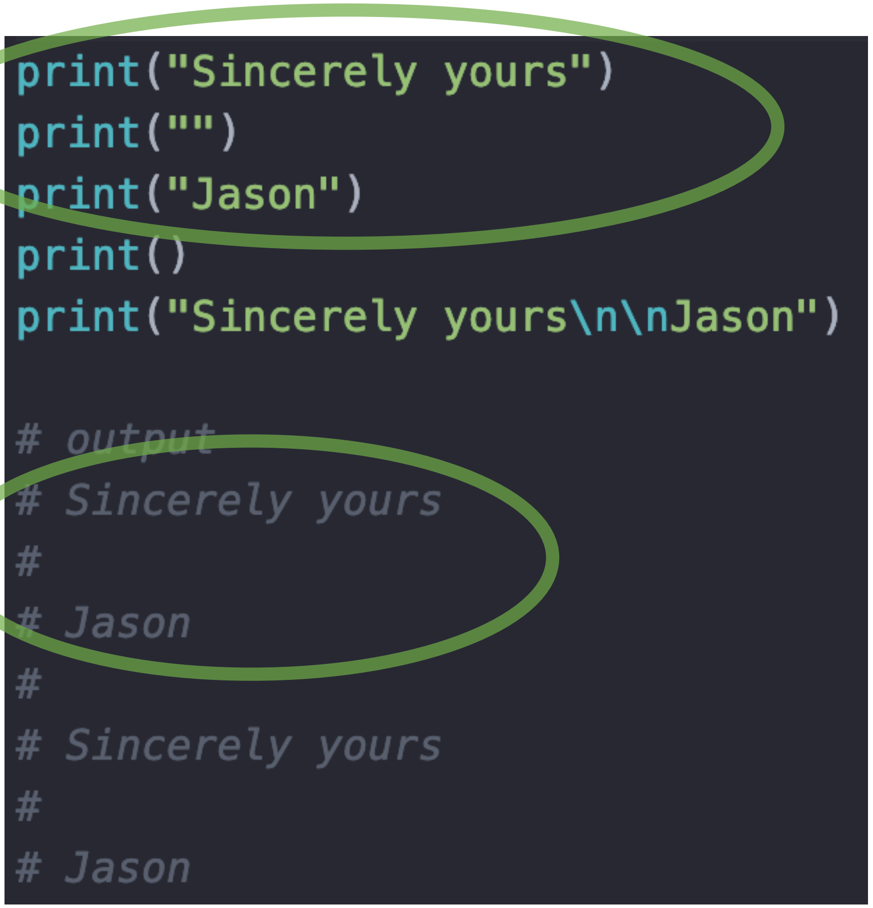
Nice output with a blank line between the text



```
print("Sincerely yours")
print("")
print("Jason")
print()
print("Sincerely yours\n\nJason")

# output
# Sincerely yours
#
# Jason
#
# Sincerely yours
#
# Jason
```

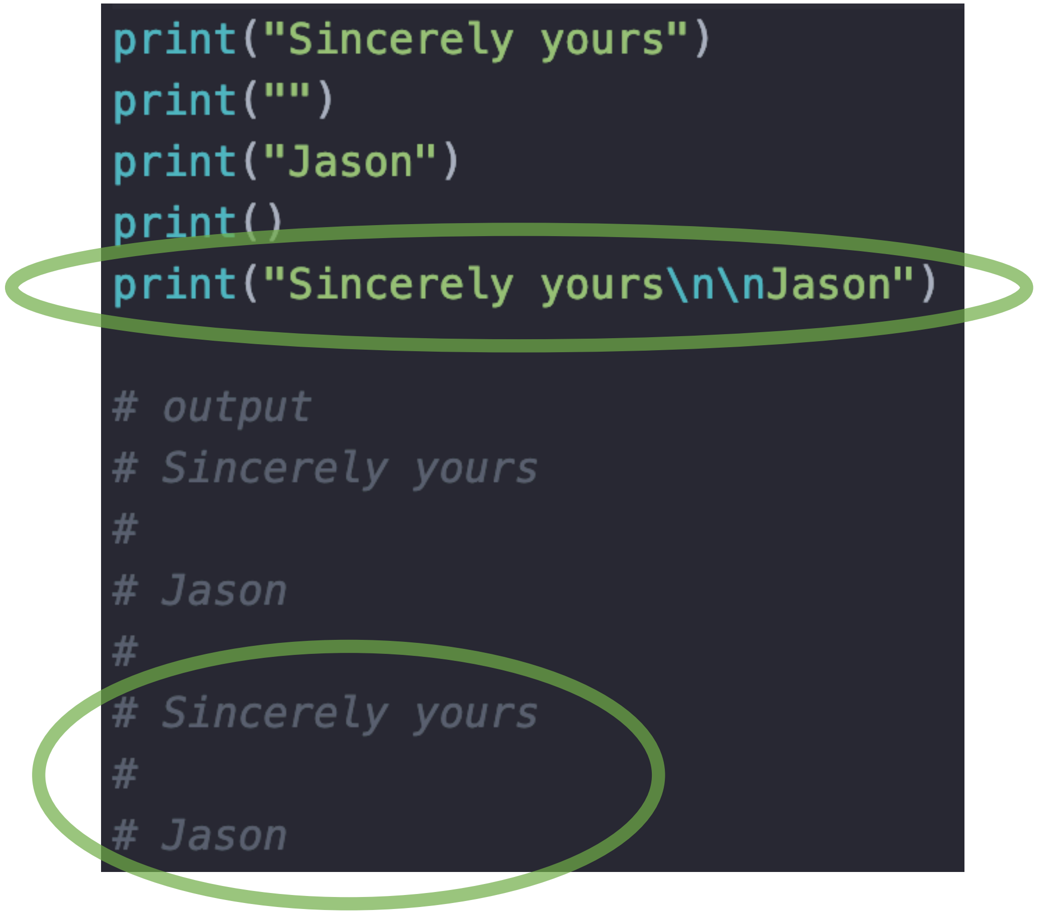
String Formatting

This can be created with 3 separate print statements


```
print("Sincerely yours")
print("")
print("Jason")
print()
print("Sincerely yours\n\nJason")

# output
# Sincerely yours
#
# Jason
#
# Sincerely yours
#
# Jason
```

String Formatting

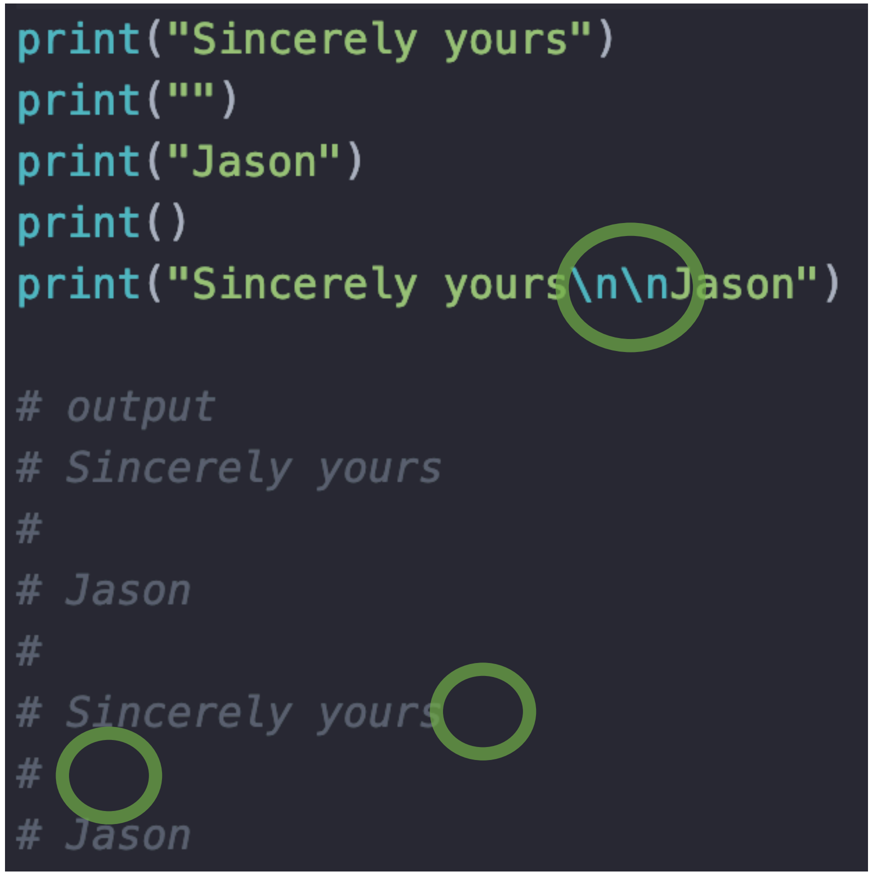
Or with a single print statement containing 2 \n characters



```
print("Sincerely yours")
print("")
print("Jason")
print()
print("Sincerely yours\n\nJason")

# output
# Sincerely yours
#
# Jason
#
# Sincerely yours
#
# Jason
```

String Formatting

Or with a single print statement containing 2 \n characters



```
print("Sincerely yours")
print("")
print("Jason")
print()
print("Sincerely yours\n\nJason")

# output
# Sincerely yours
#
# Jason
#
# Sincerely yours
#
# Jason
```

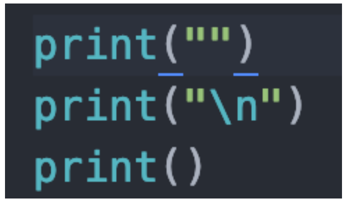
String Formatting

We have now seen 3 ways of printing a blank line

String Formatting

We have now seen 3 ways of printing a blank line

Pass an empty string “” to print()

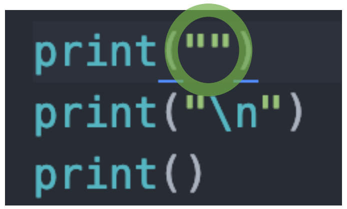


```
print("")
print("\n")
print()
```

String Formatting

We have now seen 3 ways of printing a blank line

Pass an empty string “” to print()



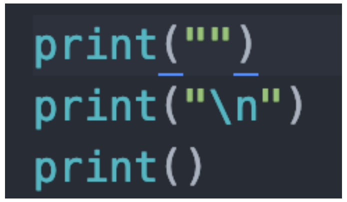
```
print("")
print("\n")
print()
```

String Formatting

We have now seen 3 ways of printing a blank line

Pass an empty string “” to print()

Pass an newline character \n to print()



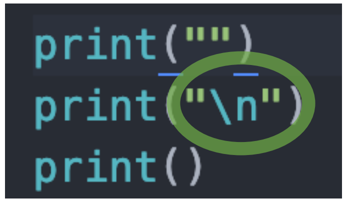
```
print("")
print("\n")
print()
```

String Formatting

We have now seen 3 ways of printing a blank line

Pass an empty string “” to print()

Pass an newline character \n to print()



```
print("")
print("\n")
print()
```

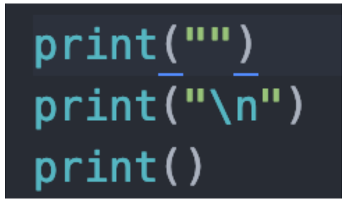
String Formatting

We have now seen 3 ways of printing a blank line

Pass an empty string “” to print()

Pass an newline character \n to print()

Pass nothing to print()



```
print("")
print("\n")
print()
```

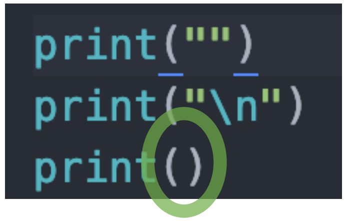
String Formatting

We have now seen 3 ways of printing a blank line

Pass an empty string “” to print()

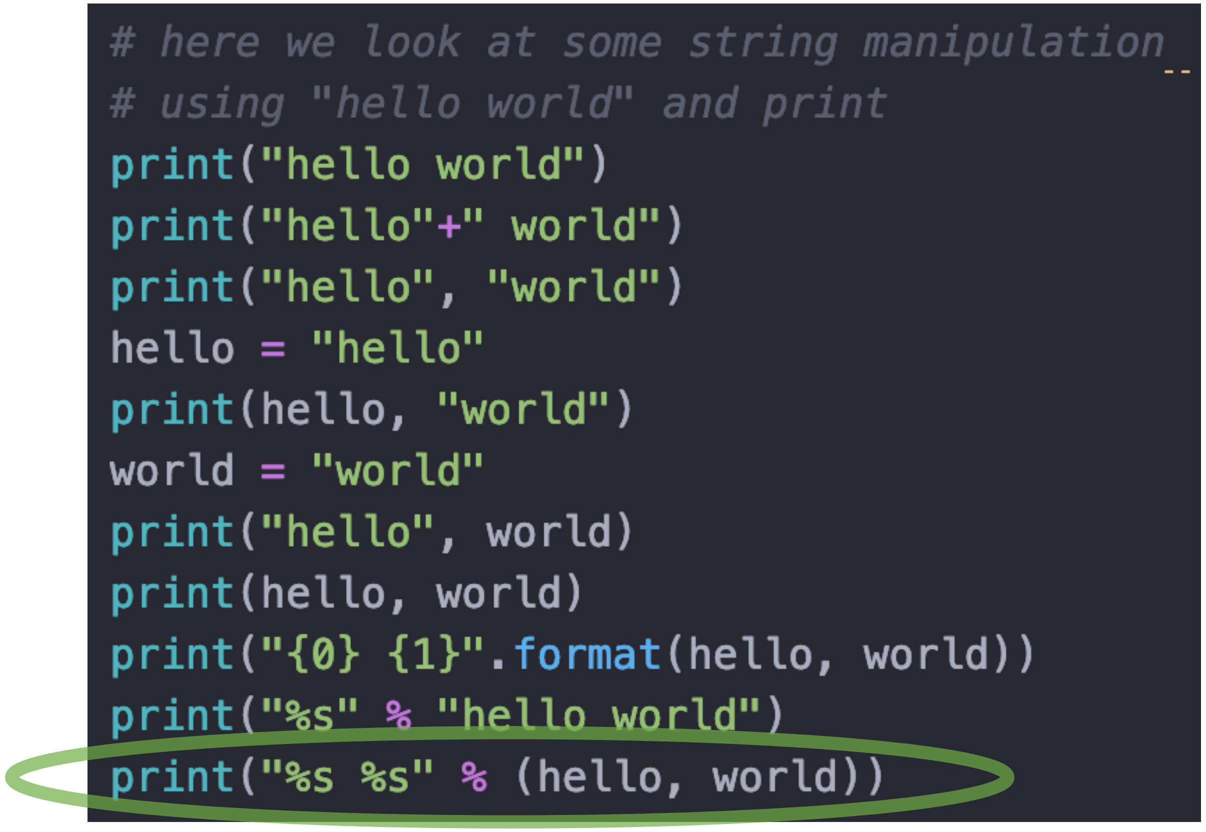
Pass an newline character \n to print()

Pass nothing to print()



```
print("")
print("\n")
print()
```

String Manipulation



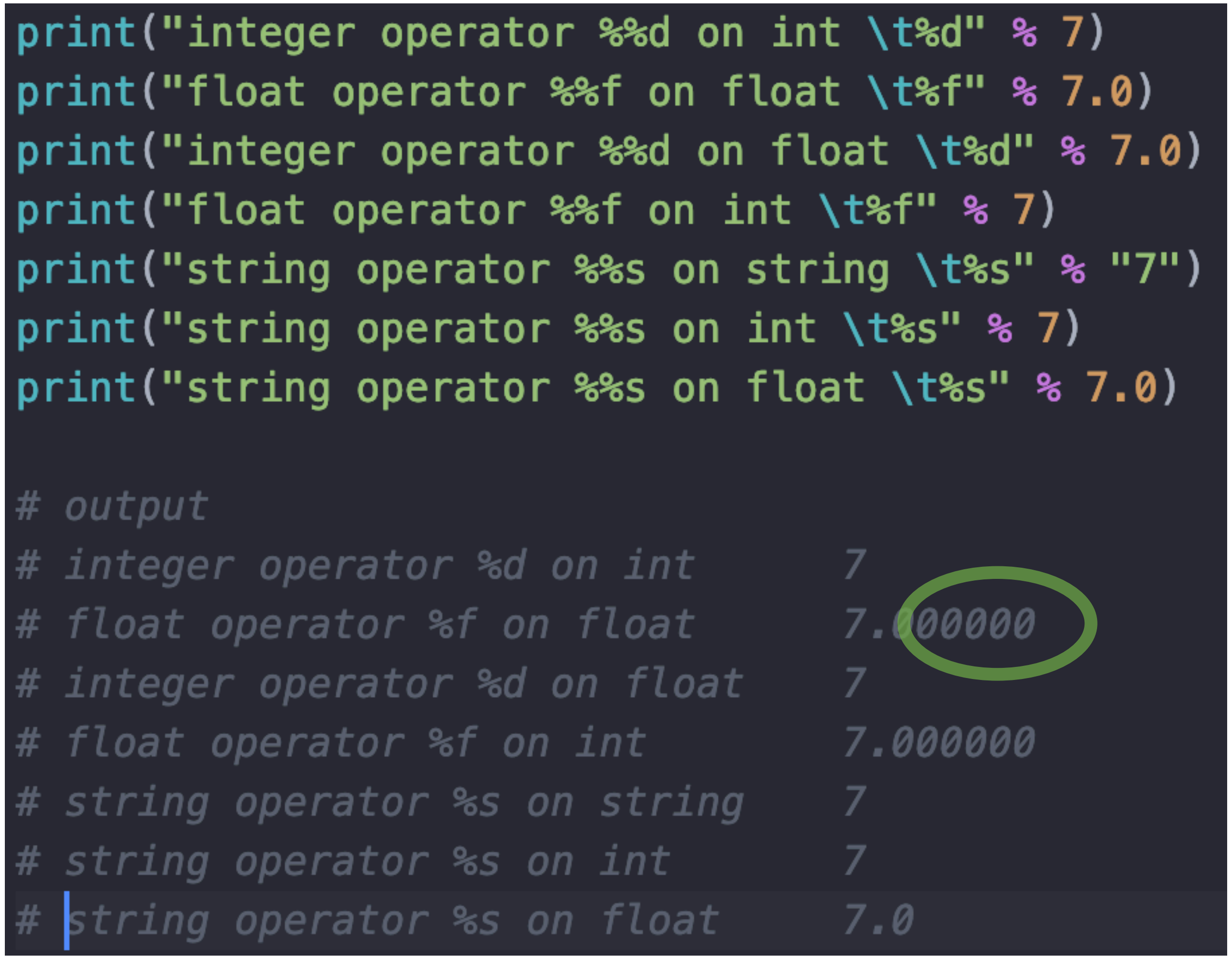
```python
# here we look at some string manipulation
# using "hello world" and print
print("hello world")
print("hello"+" world")
print("hello", "world")
hello = "hello"
print(hello, "world")
world = "world"
print("hello", world)
print(hello, world)
print("{0} {1}".format(hello, world))
print("%s" % "hello world")
print("%s %s" % (hello, world))
```

Print(“hello world”) – format the string variables hello and world using two %s operators

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Formatting

One last formatting issue – modifying the number of decimal places



```
print("integer operator %%d on int \t%d" % 7)
print("float operator %%f on float \t%f" % 7.0)
print("integer operator %%d on float \t%d" % 7.0)
print("float operator %%f on int \t%f" % 7)
print("string operator %%s on string \t%s" % "7")
print("string operator %%s on int \t%s" % 7)
print("string operator %%s on float \t%s" % 7.0)

# output
# integer operator %d on int      7
# float operator %f on float      7.000000
# integer operator %d on float    7
# float operator %f on int        7.000000
# string operator %s on string    7
# string operator %s on int       7
# string operator %s on float     7.0
```

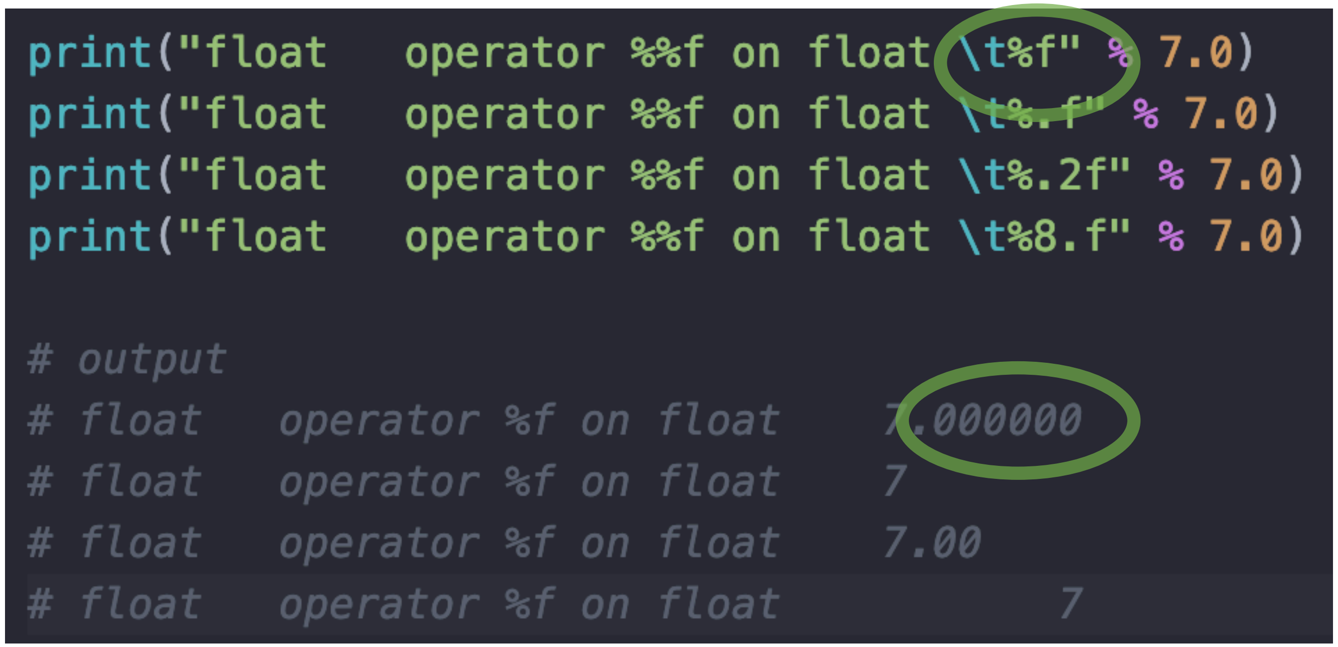
String Formatting

String formatting offers a mechanism to add spacing and reduce decimal places

String Formatting

String formatting offers a mechanism to add spacing and reduce decimal places

%f prints the number plus 6 decimal places

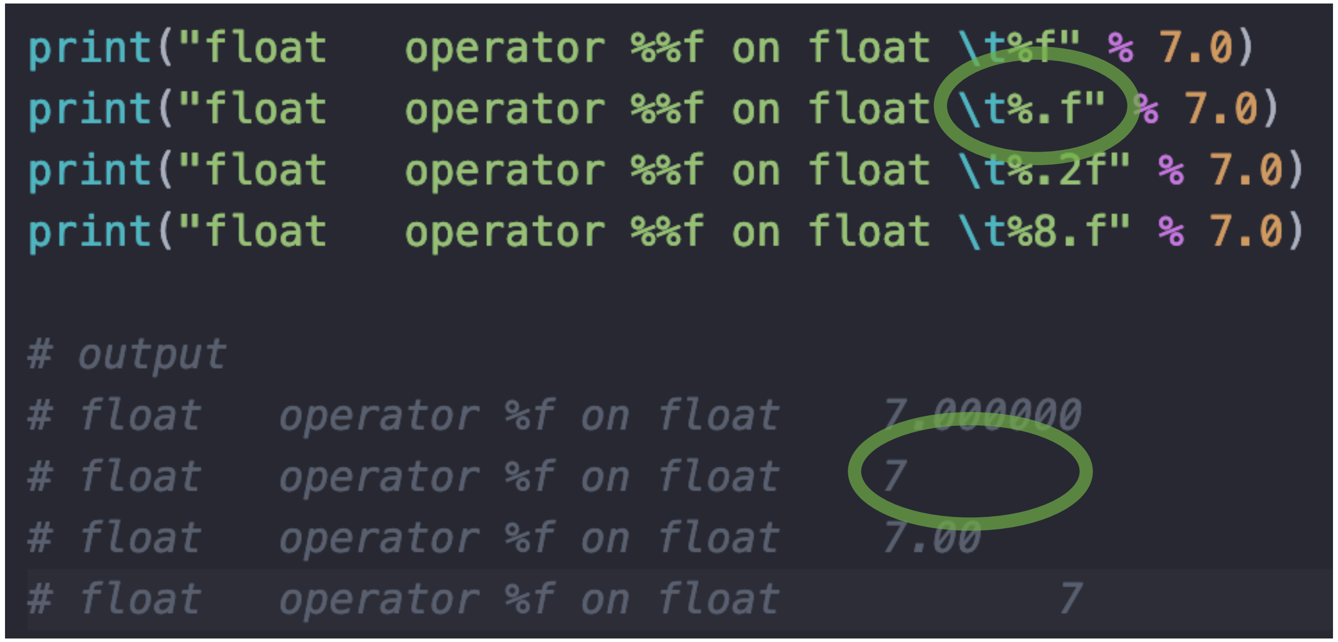


```
print("float   operator %%f on float \t%f" % 7.0)
print("float   operator %%f on float \t%.f" % 7.0)
print("float   operator %%f on float \t%.2f" % 7.0)
print("float   operator %%f on float \t%8.f" % 7.0)

# output
# float   operator %f on float     7.000000
# float   operator %f on float     7
# float   operator %f on float     7.00
# float   operator %f on float            7
```

String Formatting

String formatting offers a mechanism to add spacing and reduce decimal places

%.f prints the number and no decimal places

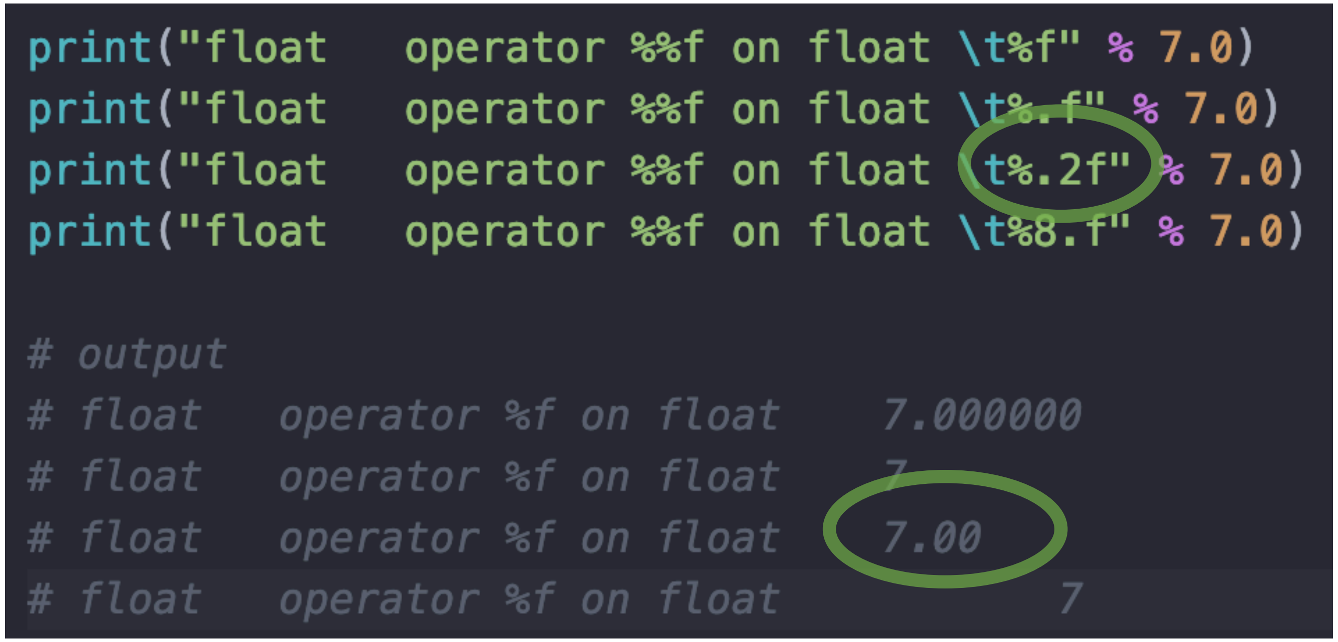


```
print("float   operator %%f on float \t%f" % 7.0)
print("float   operator %%f on float \t%.f" % 7.0)
print("float   operator %%f on float \t%.2f" % 7.0)
print("float   operator %%f on float \t%8.f" % 7.0)

# output
# float   operator %f on float    7.000000
# float   operator %f on float    7
# float   operator %f on float    7.00
# float   operator %f on float           7
```

String Formatting

String formatting offers a mechanism to add spacing and reduce decimal places

%.2f prints the number plus 2 decimal places

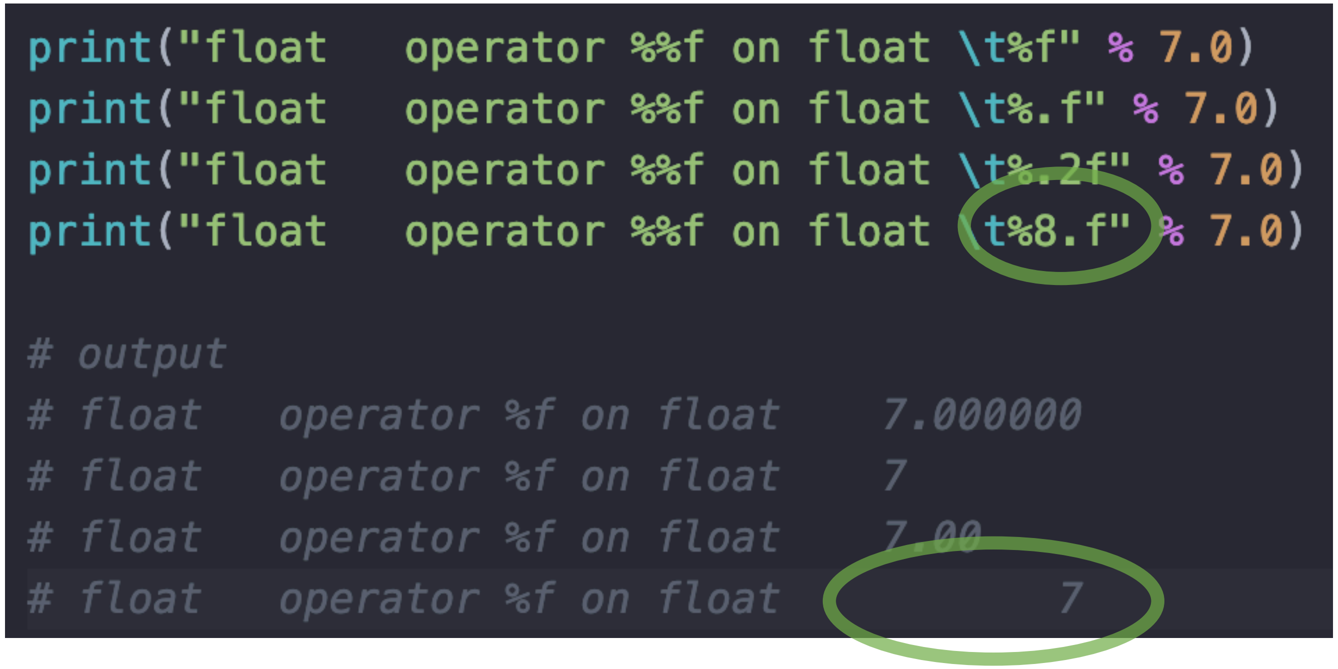


```
print("float   operator %%f on float \t%f" % 7.0)
print("float   operator %%f on float \t%.f" % 7.0)
print("float   operator %%f on float \t%.2f" % 7.0)
print("float   operator %%f on float \t%8.f" % 7.0)

# output
# float   operator %f on float    7.000000
# float   operator %f on float    7
# float   operator %f on float    7.00
# float   operator %f on float           7
```

String Formatting

String formatting offers a mechanism to add spacing and reduce decimal places

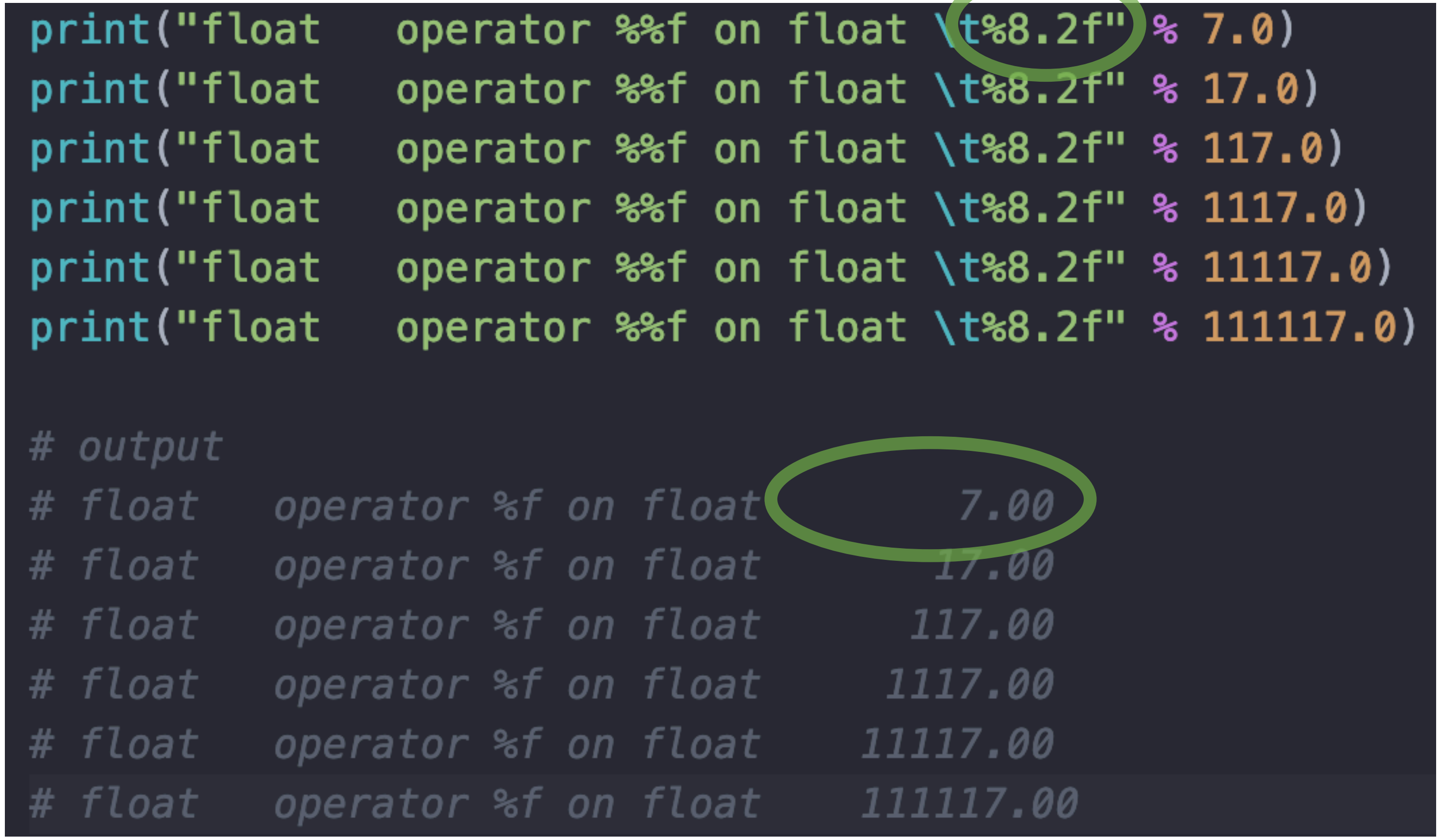
%8.f prints 8 characters - the number, no decimal places and 7 preceding blank spaces



```
print("float   operator %%f on float \t%f" % 7.0)
print("float   operator %%f on float \t%.f" % 7.0)
print("float   operator %%f on float \t%.2f" % 7.0)
print("float   operator %%f on float \t%8.f" % 7.0)

# output
# float   operator %f on float    7.000000
# float   operator %f on float    7
# float   operator %f on float    7.00
# float   operator %f on float           7
```

String Formatting

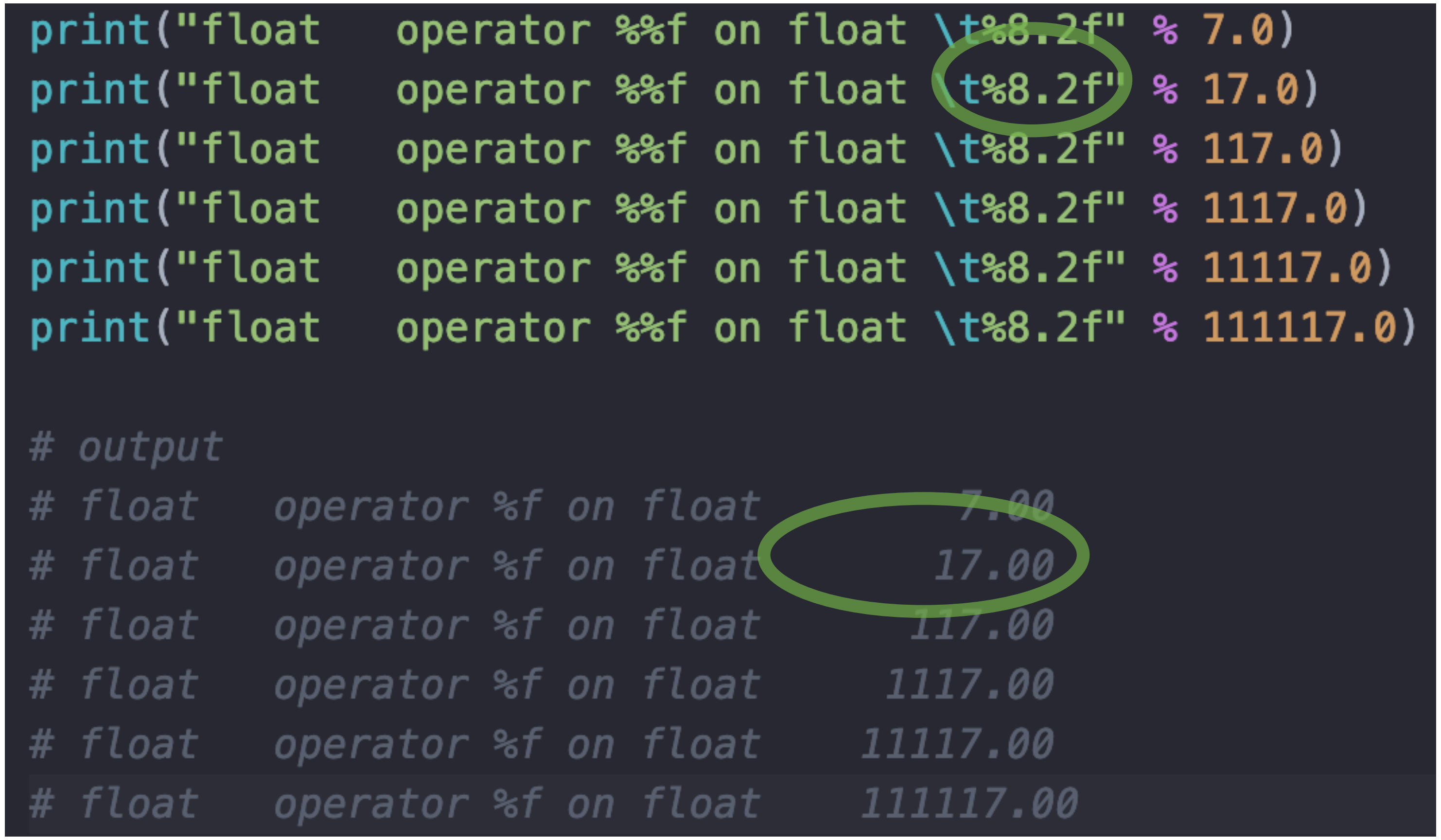
%8.2f for 7.0 prints 8 characters - the number, a decimal point, two decimal places and 4 preceding blank spaces



```
print("float   operator %%f on float \t%8.2f" % 7.0)
print("float   operator %%f on float \t%8.2f" % 17.0)
print("float   operator %%f on float \t%8.2f" % 117.0)
print("float   operator %%f on float \t%8.2f" % 1117.0)
print("float   operator %%f on float \t%8.2f" % 11117.0)
print("float   operator %%f on float \t%8.2f" % 111117.0)


# output
# float   operator %f on float        7.00
# float   operator %f on float       17.00
# float   operator %f on float      117.00
# float   operator %f on float     1117.00
# float   operator %f on float    11117.00
# float   operator %f on float    111117.00
```

String Formatting

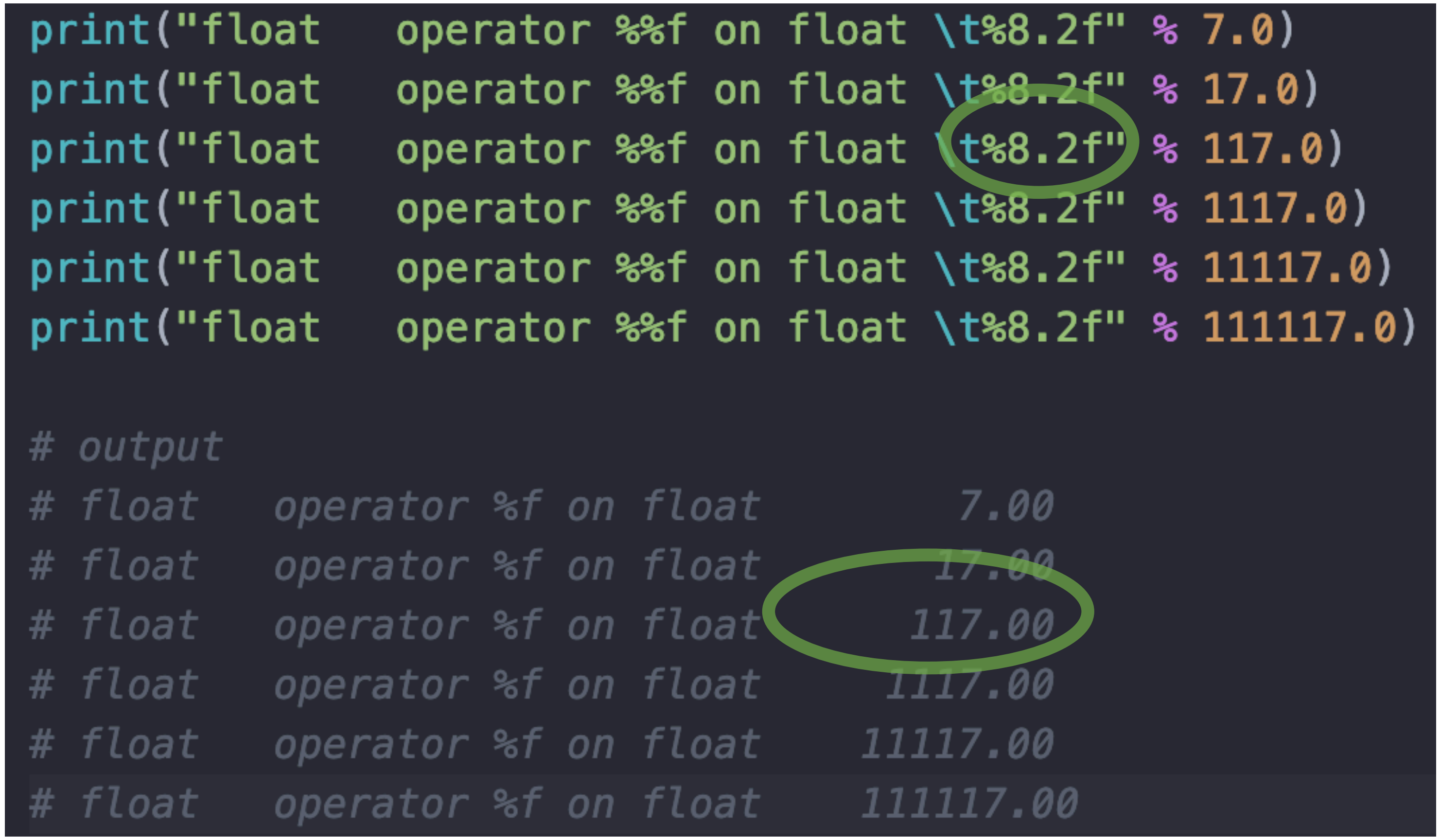
%8.2f for 17.0 prints 8 characters - the number, a decimal point, two decimal places and 3 preceding blank spaces



```python
print("float   operator %%f on float \t%8.2f" % 7.0)
print("float   operator %%f on float \t%8.2f" % 17.0)
print("float   operator %%f on float \t%8.2f" % 117.0)
print("float   operator %%f on float \t%8.2f" % 1117.0)
print("float   operator %%f on float \t%8.2f" % 11117.0)
print("float   operator %%f on float \t%8.2f" % 111117.0)

# output
# float   operator %f on float      7.00
# float   operator %f on float     17.00
# float   operator %f on float    117.00
# float   operator %f on float   1117.00
# float   operator %f on float  11117.00
# float   operator %f on float  111117.00
```

String Formatting

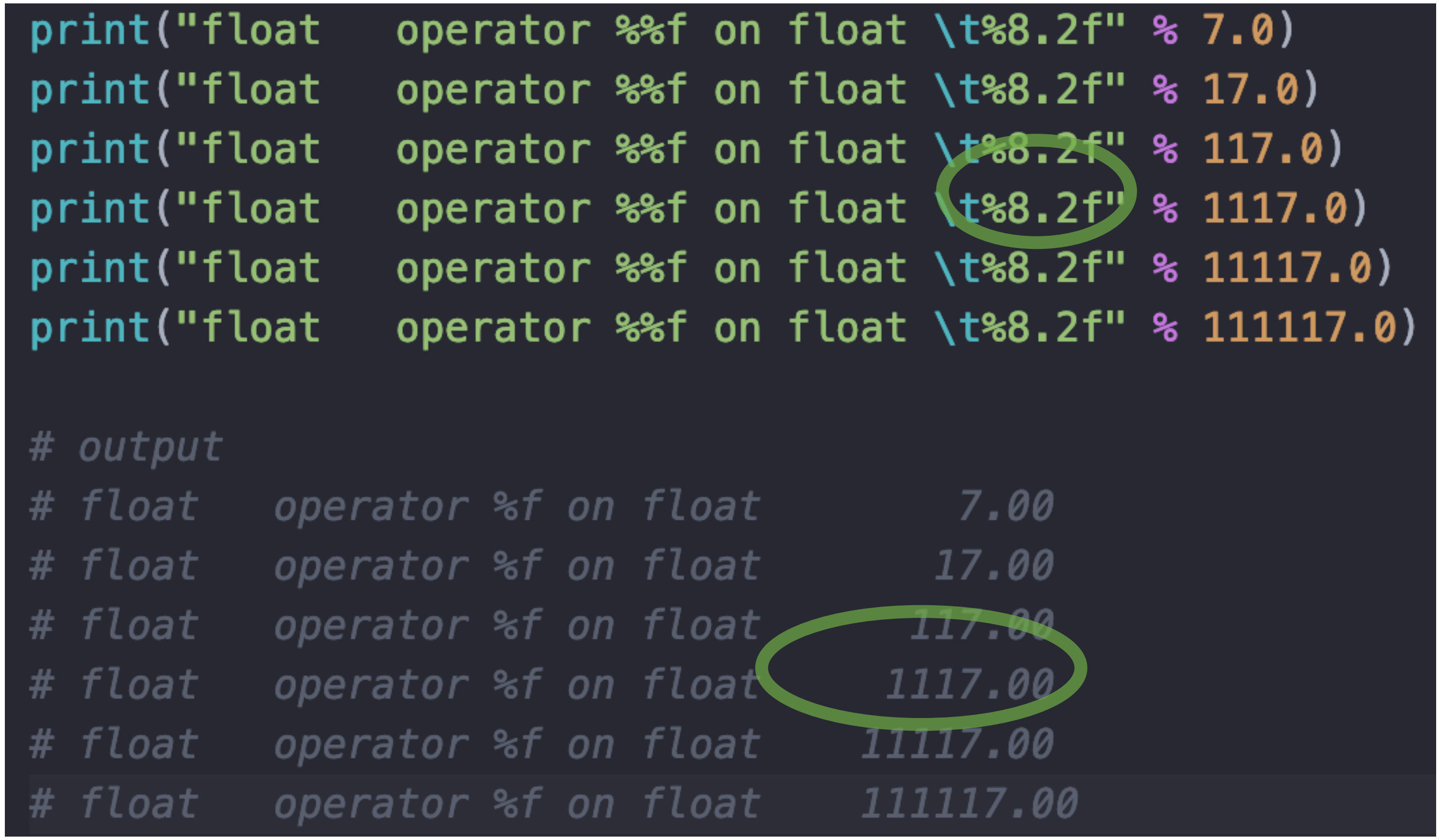
%8.2f for 117.0 prints 8 characters - the number, a decimal point, two decimal places and 2 preceding blank spaces



```
print("float   operator %%f on float \t%8.2f" % 7.0)
print("float   operator %%f on float \t%8.2f" % 17.0)
print("float   operator %%f on float \t%8.2f" % 117.0)
print("float   operator %%f on float \t%8.2f" % 1117.0)
print("float   operator %%f on float \t%8.2f" % 11117.0)
print("float   operator %%f on float \t%8.2f" % 111117.0)

# output
# float   operator %f on float        7.00
# float   operator %f on float       17.00
# float   operator %f on float      117.00
# float   operator %f on float     1117.00
# float   operator %f on float    11117.00
# float   operator %f on float    111117.00
```

String Formatting

%8.2f for 1117.0 prints 8 characters - the number, a decimal point, two decimal places and 1 preceding blank spaces

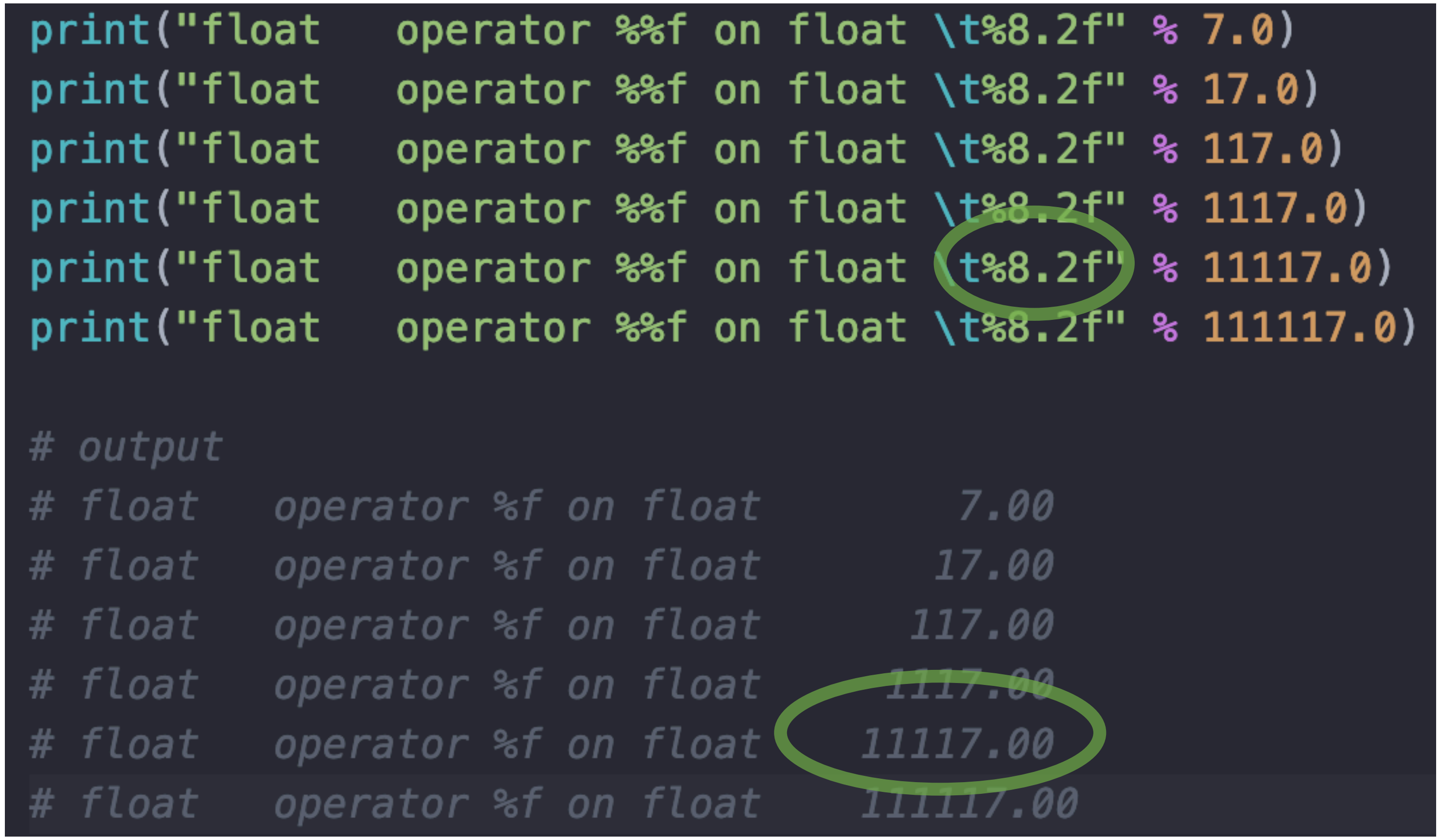


```
print("float   operator %%f on float \t%8.2f" % 7.0)
print("float   operator %%f on float \t%8.2f" % 17.0)
print("float   operator %%f on float \t%8.2f" % 117.0)
print("float   operator %%f on float \t%8.2f" % 1117.0)
print("float   operator %%f on float \t%8.2f" % 11117.0)
print("float   operator %%f on float \t%8.2f" % 111117.0)


# output
# float   operator %f on float        7.00
# float   operator %f on float       17.00
# float   operator %f on float      117.00
# float   operator %f on float     1117.00
# float   operator %f on float    11117.00
# float   operator %f on float    111117.00
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String Formatting

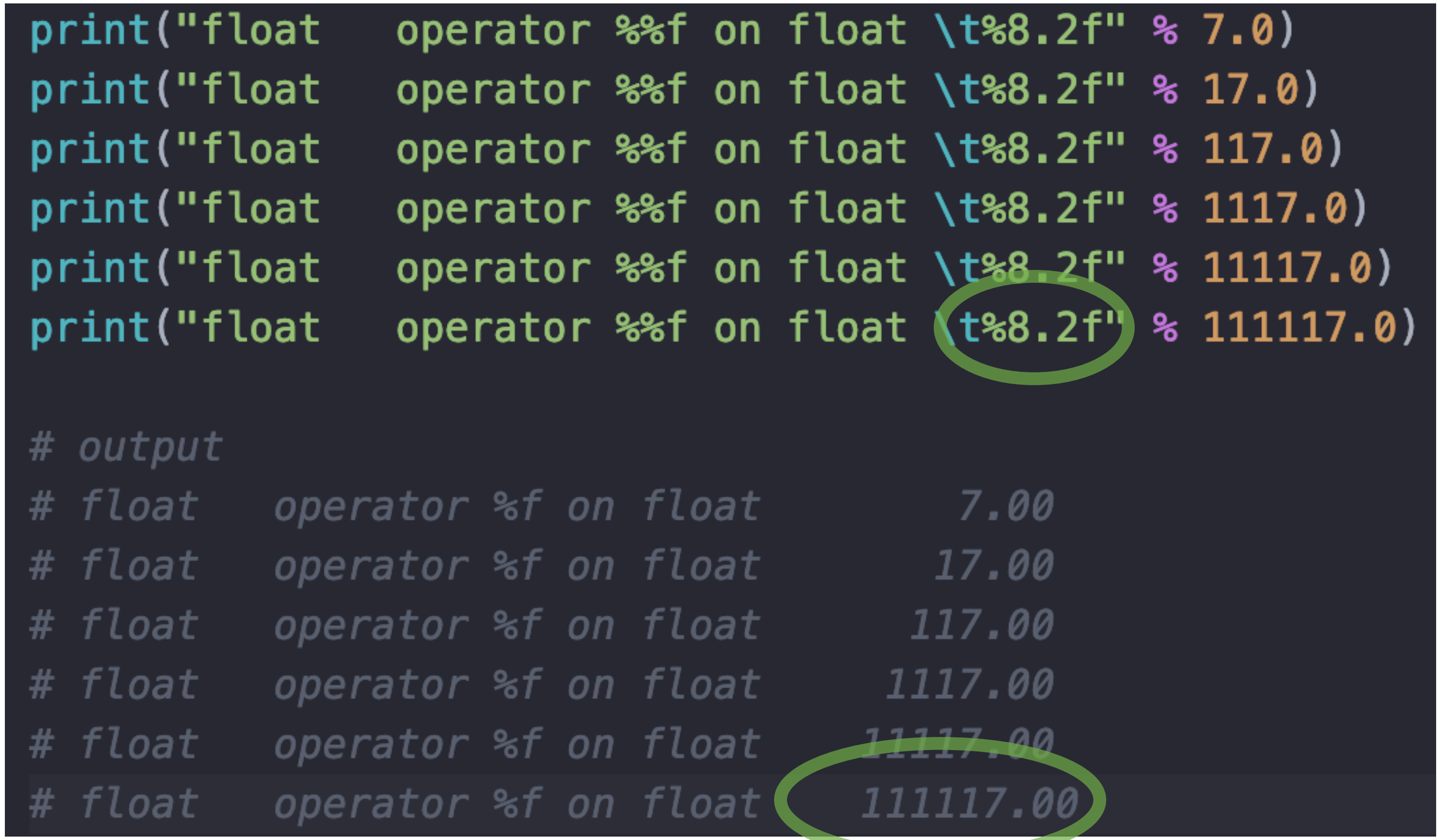
%8.2f for 11117.0 prints 8 characters - the number, a decimal point, two decimal places and no preceding blank spaces



```
print("float    operator %%f on float \t%8.2f" % 7.0)
print("float    operator %%f on float \t%8.2f" % 17.0)
print("float    operator %%f on float \t%8.2f" % 117.0)
print("float    operator %%f on float \t%8.2f" % 1117.0)
print("float    operator %%f on float \t%8.2f" % 11117.0)
print("float    operator %%f on float \t%8.2f" % 111117.0)


# output
# float    operator %f on float         7.00
# float    operator %f on float        17.00
# float    operator %f on float       117.00
# float    operator %f on float      1117.00
# float    operator %f on float     11117.00
# float    operator %f on float     111117.00
```

String Formatting

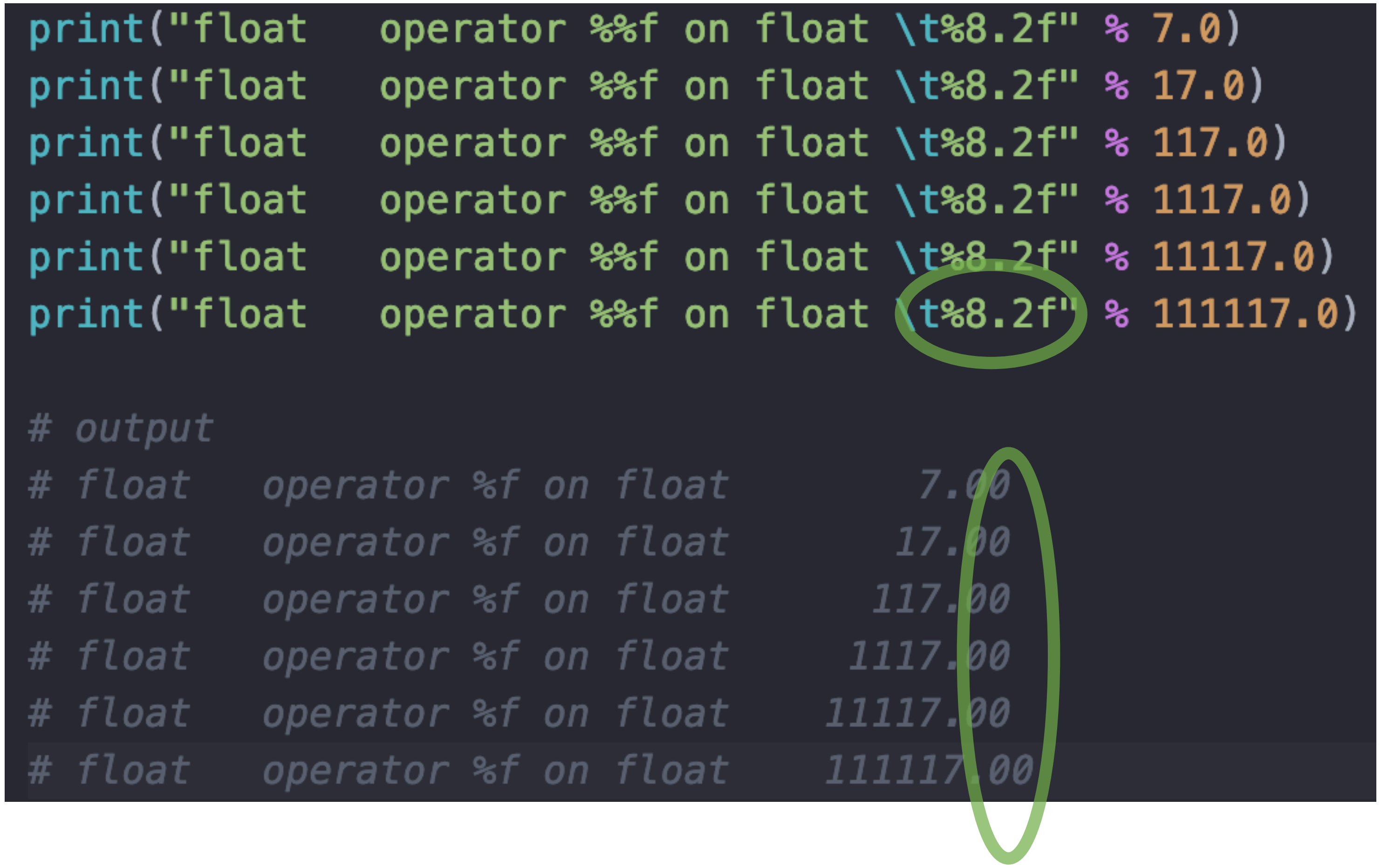
%8.2f for 111117.0 prints 8 characters - the number, a decimal point two decimal places and no preceding blank spaces



```
print("float    operator %%f on float \t%8.2f" % 7.0)
print("float    operator %%f on float \t%8.2f" % 17.0)
print("float    operator %%f on float \t%8.2f" % 117.0)
print("float    operator %%f on float \t%8.2f" % 1117.0)
print("float    operator %%f on float \t%8.2f" % 11117.0)
print("float    operator %%f on float \t%8.2f" % 111117.0)

# output
# float    operator %f on float        7.00
# float    operator %f on float       17.00
# float    operator %f on float      117.00
# float    operator %f on float     1117.00
# float    operator %f on float    11117.00
# float    operator %f on float    111117.00
```

String Formatting

%8.2f for 111117.0 prints 8 characters – but it is no longer formatting properly



```
print("float   operator %%f on float \t%8.2f" % 7.0)
print("float   operator %%f on float \t%8.2f" % 17.0)
print("float   operator %%f on float \t%8.2f" % 117.0)
print("float   operator %%f on float \t%8.2f" % 1117.0)
print("float   operator %%f on float \t%8.2f" % 11117.0)
print("float   operator %%f on float \t%8.2f" % 111117.0)

# output
# float   operator %f on float        7.00
# float   operator %f on float       17.00
# float   operator %f on float      117.00
# float   operator %f on float     1117.00
# float   operator %f on float    11117.00
# float   operator %f on float    111117.00
```

String Formatting

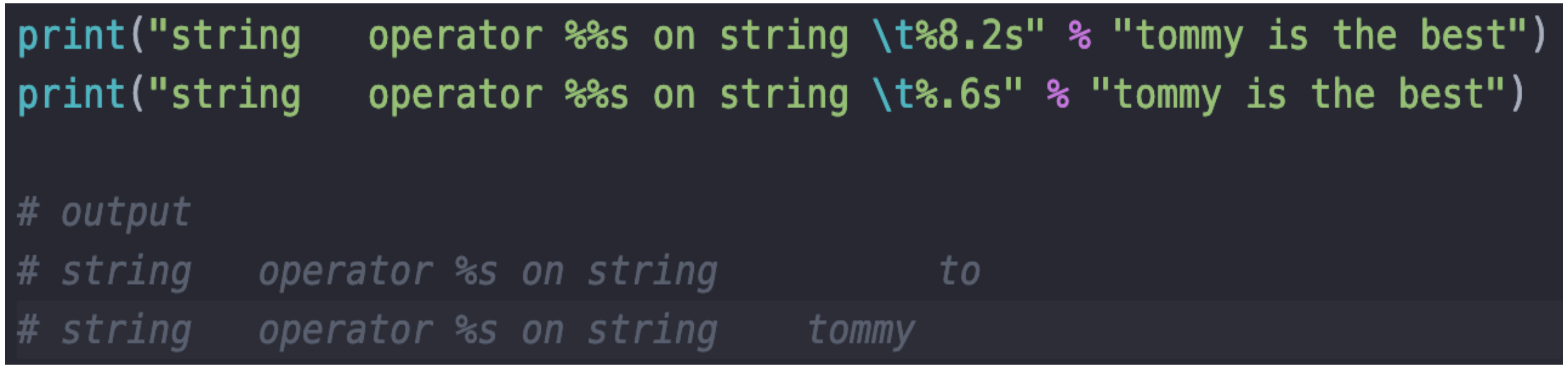
Does this work for the other formatting operators?

Lets try string - %s

String Formatting

Does this work for the other formatting operators?

Lets try string - %s



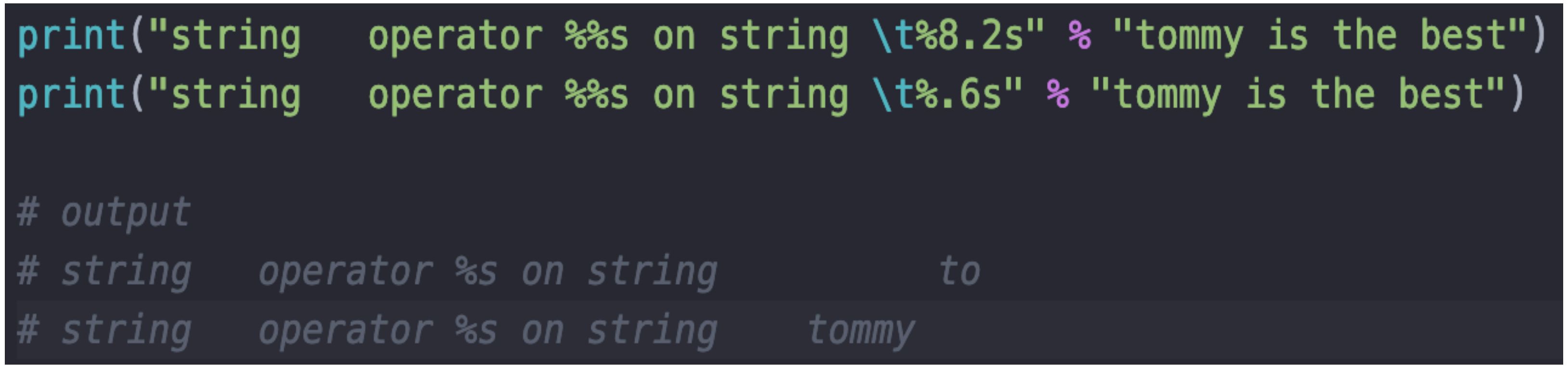
```
print("string   operator %%s on string \t%8.2s" % "tommy is the best")
print("string   operator %%s on string \t%.6s" % "tommy is the best")

# output
# string   operator %s on string          to
# string   operator %s on string     tommy
```

String Formatting

Does this work for the other formatting operators?

Lets try string - %s

When we use %8.2s – we get 8 characters, 2 from the input string and the remainder are blank



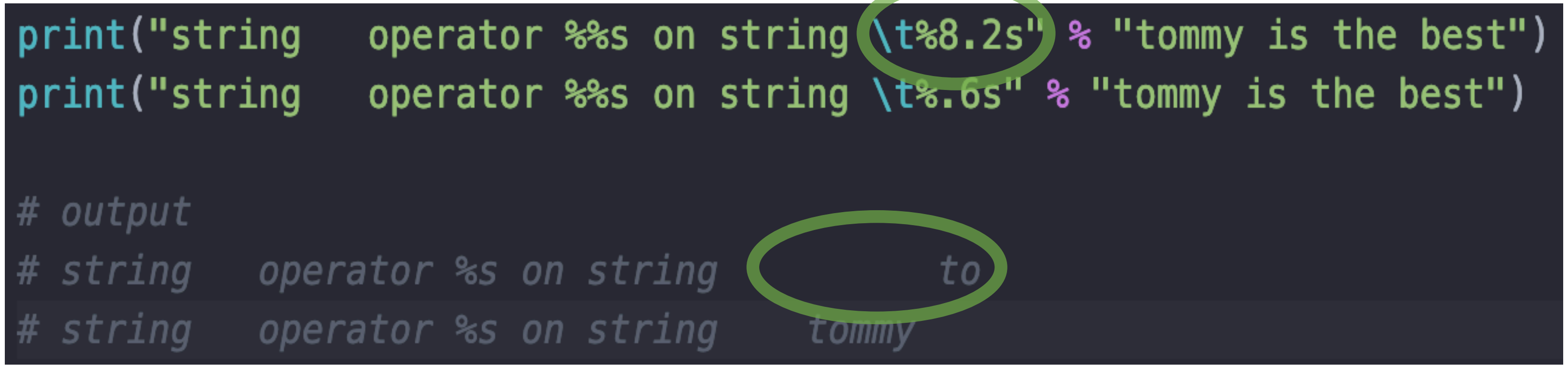
```
print("string   operator %%s on string \t%8.2s" % "tommy is the best")
print("string   operator %%s on string \t%.6s" % "tommy is the best")

# output
# string   operator %s on string          to
# string   operator %s on string    tommy
```

String Formatting

Does this work for the other formatting operators?

Lets try string - %s

When we use %8.2s – we get 8 characters, 2 from the input string and the remainder are blank spaces



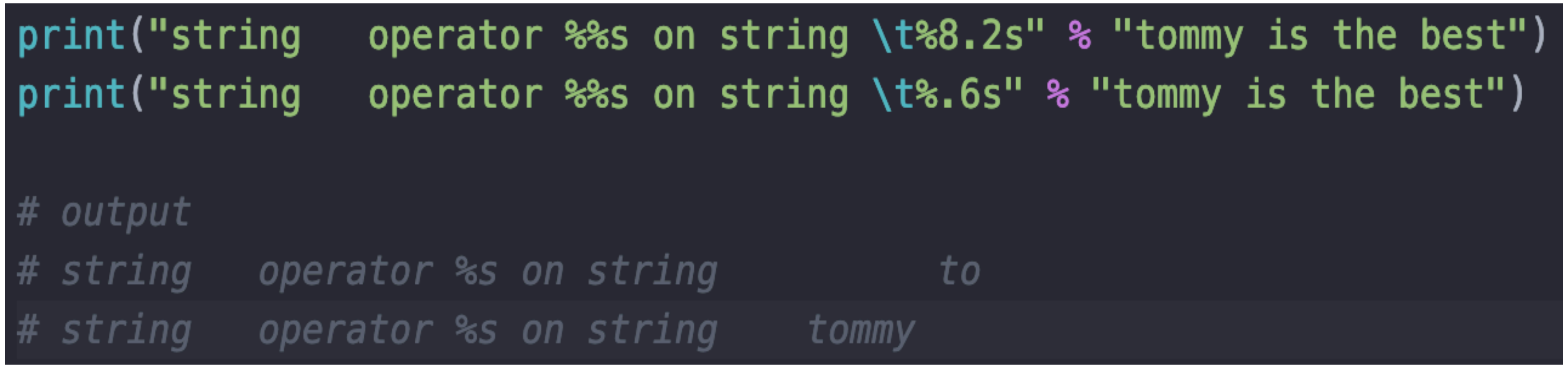
```
print("string   operator %%s on string \t%8.2s" % "tommy is the best")
print("string   operator %%s on string \t%.6s" % "tommy is the best")

# output
# string   operator %s on string         to
# string   operator %s on string     tommy
```

String Formatting

Does this work for the other formatting operators?

Lets try string - %s

When we use %.6s – we get 6 characters, 6 from the input string and no preceding blank spaces



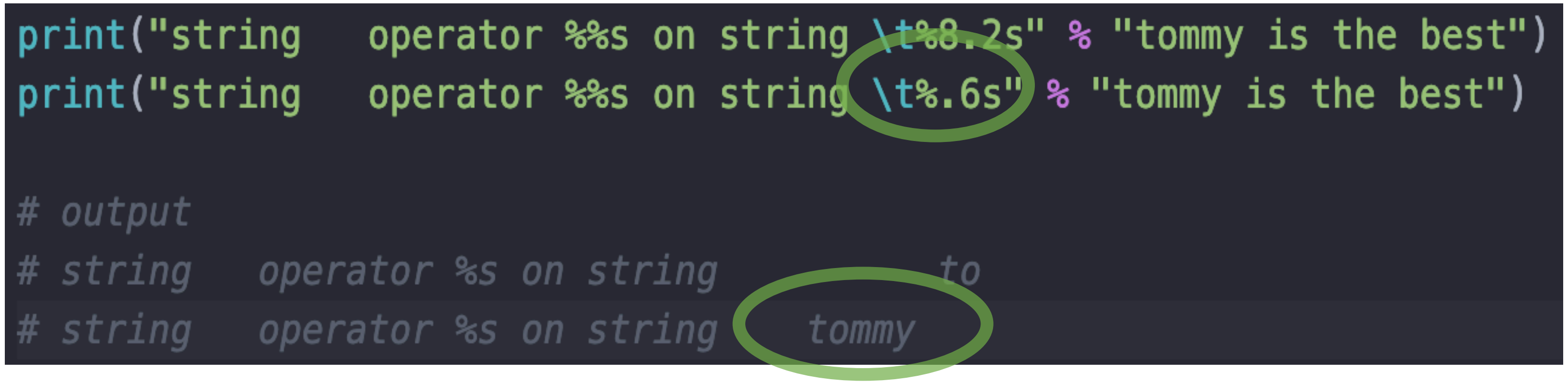
```
print("string   operator %%s on string \t%8.2s" % "tommy is the best")
print("string   operator %%s on string \t%.6s" % "tommy is the best")

# output
# string   operator %s on string          to
# string   operator %s on string     tommy
```

String Formatting

Does this work for the other formatting operators?

Lets try string - %s

When we use %.6s – we get 6 characters, 6 from the input string and no preceding blank spaces



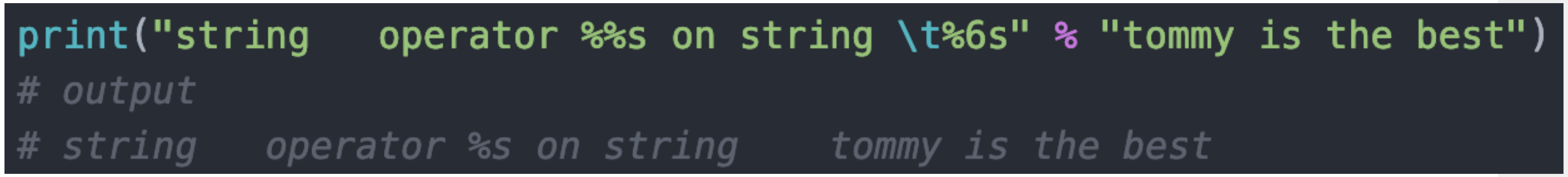
```
print("string   operator %%s on string \t%8.2s" % "tommy is the best")
print("string   operator %%s on string \t%.6s" % "tommy is the best")

# output
# string   operator %s on string          to
# string   operator %s on string    tommy
```

String Formatting

Does this work for the other formatting operators?

Lets try string - %s

When we remove the . And use %6s – the entire string is printed with no formatting



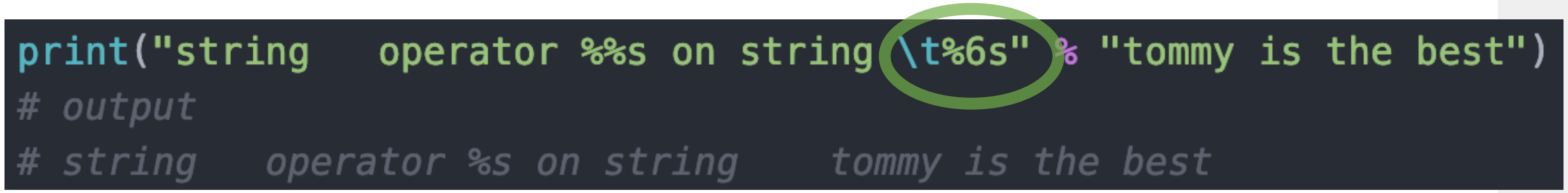
```
print("string   operator %%s on string \t%6s" % "tommy is the best")
# output
# string   operator %s on string    tommy is the best
```

String Formatting

Does this work for the other formatting operators?

Lets try string - %s

When we remove the . And use %6s – the entire string is printed with no formatting



```
print("string    operator %%s on string \t%6s" % "tommy is the best")
# output
# string    operator %s on string    tommy is the best
```

String Formatting Recap

- We looked at how to import functions from Pythons libraries
- We look at various ways of passing parameters to the print() function
- We saw that string objects have their own set of functions we can calling using the dot (.) operator
- We saw some of the % operators we can use to format input to string objects
- We saw how we can use \t (tab) and \n (newline) to modify the structure of the string output
- And we saw 3 different ways to print a blank line in Python
- Finally, we saw how to create tables in the print output, using numbers in %f

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh