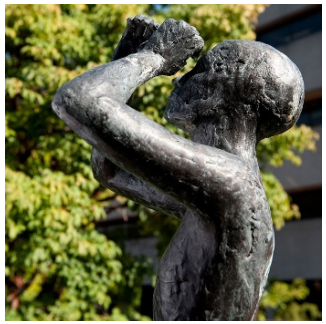


CS1117 – Introduction to Programming

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

Atom and PyCharm in G20

After yesterdays lab, I have decided to work with PyCharm for the first semester

And Atom in the second semester

Lots to learn, so let's learn the basics with one IDE first.

Announcements

Atom

If you want to install Atom on your own laptop and need help, let me know.

If enough people are interested, I'll get a room for us for about 30 minutes and we will install Atom

If only one or two are interested, we will just go to the Canteen after one of the lectures and install Atom

Announcements

Access to G20

G20 is your lab, so you have access all day every day, unless:

1. a lab/tutorial for a module you are not registered for is on
2. or like CS1117, you are not allocated to a lab on a given day at a specific time

All other times, it's your lab, so use it :)

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

print()

This is the docString for Python's print() function

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

print()

This is the docString for Python's print() function

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

First thing to note is, it is a very detailed docString and you can see immediately what extra parameters print() has and what their respective role/type() are

print()

We can see that print() has some additional parameters we can play with

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

print()

And each is detailed in the docString

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

print()

Let's look at "sep" first

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

print()

Let's look at "sep" first

sep changes how print() joins the different strings together

```
print("there is a", "in this line")
print("there is a", "in this line", sep=" - ")
print("there is a", "in this line", sep=" word ")

# output
# there is a in this line
# there is a - in this line
# there is a word in this line
```

print()

Let's look at "sep" first

sep changes how print() joins the different strings together

```
print("there is a", "in this line")
print("there is a", "in this line", sep=" - ")
print("there is a", "in this line", sep=" word ")

# output
# there is a in this line
# there is a - in this line
# there is a word in this line
```

Default sep adds a space between the individual strings

print()

Let's look at "sep" first

sep changes how print() joins the different strings together

```
print("there is a", "in this line")
print("there is a", "in this line", sep=" - ")
print("there is a", "in this line", sep=" word ")

# output
# there is a in this line
# there is a - in this line
# there is a word in this line
```

We can change the value of sep to " - "
this will add a dash between the individual strings

print()

Let's look at "sep" first

sep changes how print() joins the different strings together

```
print("there is a", "in this line")
print("there is a", "in this line", sep=" - ")
print("there is a", "in this line", sep=" word ")

# output
# there is a in this line
# there is a - in this line
# there is a word in this line
```

We can set the value of sep to any string we want

print()

Will sep cast for us?

If I set the value of sep to the int value 7, will sep /print() cast this to “7”?

print()

Will sep cast for us?

If I set the value of sep to the int value 7, will sep /print() cast this to “7”?

```
print("there is a", "in this line", sep=7)

# output
# Traceback (most recent call last):
#   File "./lecture_6.py", line 37, in <module>
#     print("there is a", "in this line", sep=7)
# TypeError: sep must be None or a string, not int
```

No it will not ☺

print()

This is the docString for Python's print() function

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

print()

Will sep cast for us?

If I set the value of to the int 7, will sep/print cast this to “7”?

```
print("there is a", "in this line", sep=7)

# output
# Traceback (most recent call last):
#   File "./lecture_6.py", line 37, in <module>
#     print("there is a", "in this line", sep=7)
# TypeError: sep must be None or a string, not int
```

No it will not ☺

print()

But we can use string formatting with sep

```
print("each", "word", "in", "this", "sentence",
      "is ", "on", "a", "different", "line", sep="\n")

# output
# each
# word
# in
# this
# sentence
# is
# on
# a
# different
# line
```

print()

But we can use string formatting with sep

```
print("each", "word", "in", "this", "sentence",
      "is ", "on", "a", "different", "line", sep="\n")

# output
# each
# word
# in
# this
# sentence
# is
# on
# a
# different
# line
```

print()

But we can use string formatting with sep

```
print("each", "word", "in", "this", "sentence",
      "is ", "on", "a", "different", "line", sep="\n")

# output
# each
# word
# in
# this
# sentence
# is
# on
# a
# different
# line
```

print()

We have already seen how print adds a return carriage or “newline” (\n) at the end of each print()

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

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

print()

But print() gives us an option to change this, using end

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

print()

Here we set the value of end to equal the empty string

```
print("This line stops here")
print("This line runs over ", end="")
print("two lines")

# output
# This line stops here
# This line runs over two lines
```

print()

Here we set the value of end to equal the empty string

```
print("This line stops here")
print("This line runs over ", end="")
print("two lines")

# output
# This line stops here
# This line runs over two lines
```

So two print statements can be output on the one line

print()

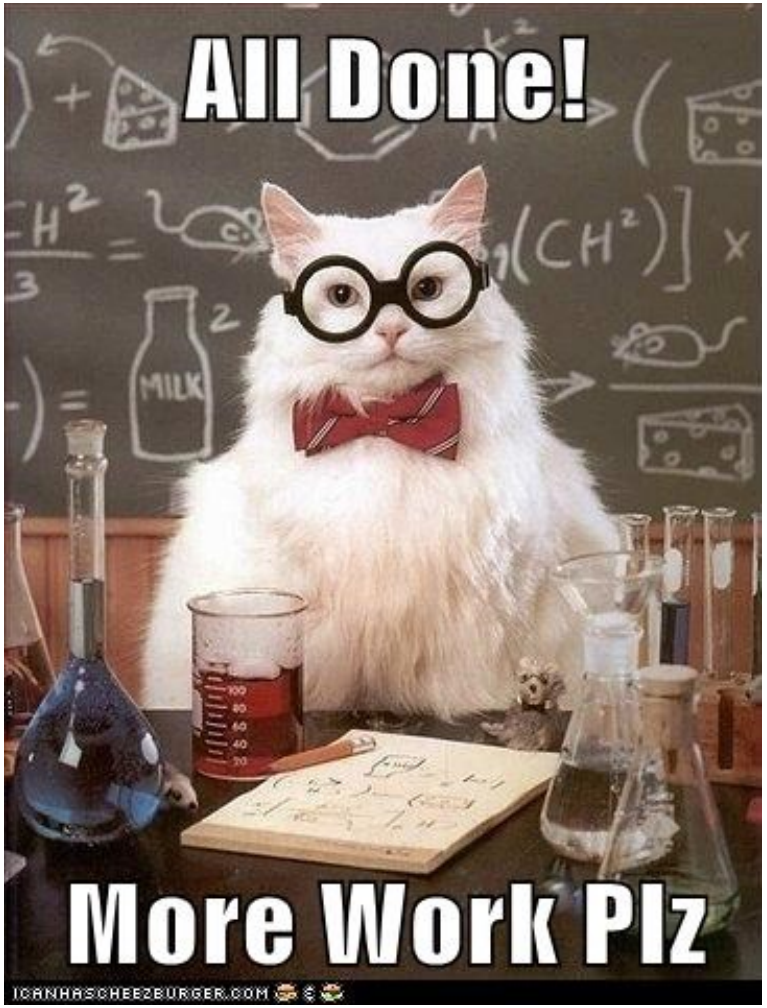
So going forward consider:

1. The content of your docString
   - Explain each parameter, making it clear what is expected with reference to type and content
2. If your docString states parameter 2 expects an int
   - Then cast parameter 2 to an int
   - input_2 = int(param_2)
   - If this causes a run time issue that is on the user ☺
   - Your code works as defined in your docString
3. Where possible consider extra parameters
   - If you have print statements in your function, then maybe something like sep or end might be nice to pass through to print
   - def my_func(my_param, my_sep=' ', my_end='\n'):

     print(my_param, sep=my_sep, end=my_end)

Canvas Student App

Let’s Sign into this lecture now



Import

Previously we looked at import as a mechanism for getting access to Pythons in-built library of functions

```
import statistics
```

```
from statistics import mean
```

But we can use import to re-use function we have created

Import

Here is our “average_of_two()” function stored in “lecture_5.py”

lecture_5.py

```
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", 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))
```

Import

I create a new python file, called “lecture_6.py”
And I want to call “average_of_two()”

lecture_5.py | lecture_6.py

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

# output
# Traceback (most recent call last):
#   File "./lecture_6.py", line 5, in <module>
#     print(average_of_two(number_one, number_two))
# NameError: name 'average_of_two' is not defined
```

Import

I create a new python file, called “lecture_6.py”
And I want to call “average_of_two()”

lecture_5.py | lecture_6.py

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

# output
# Traceback (most recent call last):
#   File "./lecture_6.py", line 5, in <module>
#     print(average_of_two(number_one, number_two))
# NameError: name 'average_of_two' is not defined
```

Import

I create a new python file, called “lecture_6.py”
And I want to call “average_of_two()”

lecture_5.py | lecture_6.py

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

# output
# Traceback (most recent call last):
#   File "./lecture_6.py", line 5, in <module>
#     print(average_of_two(number_one, number_two))
# NameError: name 'average_of_two' is not defined
```

You see Atom is telling me I have a problem – red underline

Import

I create a new python file, called “lecture_6.py”
And I want to call “average_of_two()”

lecture_5.py | lecture_6.py

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

# output
# Traceback (most recent call last):
#   File "./lecture_6.py", line 5, in <module>
#     print(average_of_two(number_one, number_two))
# NameError: name 'average_of_two' is not defined
```

And the output is telling me the function is not defined

Import

But if I add “import lecture_5” to the top of my new file

lecture_5.py | lecture_6.py

```
import lecture_5

number_one = 2
number_two = 4
print(lecture_5.average_of_two(number_one, number_two))

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

Import

And I call the function using the dot operator “lecture_5.<function>”

lecture_5.py | lecture_6.py

```
import lecture_5

number_one = 2
number_two = 4
print(lecture_5.average_of_two(number_one, number_two))

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

Import

And I call the function using the dot operator “lecture_5.<function>”

lecture_5.py | lecture_6.py

```
import lecture_5

number_one = 2
number_two = 4
print(lecture_5.average_of_two(number_one, number_two))

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

All is good ☺

Import

And I call the function using the dot operator “lecture_5.<function>”

lecture_5.py | lecture_6.py

```
import lecture_5

number_one = 2
number_two = 4
print(lecture_5.average_of_two(number_one, number_two))

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

All is good, but… the output is wrong ☹
we are seeing twice the output

Import

And I call the function using the dot operator “lecture_5.<function>”

lecture_5.py | lecture_6.py

```
import lecture_5

number_one = 2
number_two = 4
print(lecture_5.average_of_two(number_one, number_two))

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

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

This is the expected output

Import

Maybe it's how I am importing the file and I should only import that function

lecture_5.py lecture_6.py

```
from lecture_5 import average_of_two

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

Import

If I only import the function, then I do not need to use the dot operator

lecture_5.py | lecture_6.py

```
from lecture_5 import average_of_two

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

Import

If I only import the function, then I do not need to use the dot operator

lecture_5.py | lecture_6.py

```
from lecture_5 import average_of_two

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

Nope, the output is still the same ☹

Import

Let's look at our lecture_5.py file again

lecture_5.py

```
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", 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))
```

Import

Let's look at our lecture_5.py file again

lecture_5.py

```
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", 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))
```

We have the function we want

Import

Let's look at our lecture_5.py file again

lecture_5.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", 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))
```

We have the function we want, but we also have code not in a function

Import

Let's look at our lecture_5.py file again

lecture_5.py

```
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", 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))
```

And it is this code that gets called as well as our call to "average_of_two"

Import

If we change our variable values in lecture_6.py

```
from lecture_5 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  2
# number_2 is  4
# 3.0
# number_1 is  10
# number_2 is  14
# 12.0
```

Import

If we change our variable values in lecture_6.py

```
from lecture_5 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  2
# number_2 is  4
# 3.0
# number_1 is  10
# number_2 is  14
# 12.0
```

To 10 and 14, respectively

Import

If we change our variable values in lecture_6.py

```
from lecture_5 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  2
# number_2 is  4
# 3.0
# number_1 is  10
# number_2 is  14
# 12.0
```

We can see 10 and 14 are there but as the second set of print statements

Import

If we take this to it's bare minimum and only import lecture_5

```
import lecture_5


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

Import

If we take this to it's bare minimum and only import lecture_5

```
import lecture_5


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

Import

If we take this to it's bare minimum and only import lecture_5
And make no call to the "average_of_two" function

```
import lecture_5


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

Import

If we take this to it's bare minimum and only import lecture_5
And make no call to the "average_of_two" function

```
import lecture_5


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

We still have output

Import

If we take this to it's bare minimum and only import lecture_5
And make no call to the "average_of_two" function

```
import lecture_5


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

We still have output, which reflects the code not in a function in lecture_5.py

Import

If we take this to it's bare minimum and only import lecture_5
And make no call to the "average_of_two" function

```
import lecture_5


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

We still have output, which reflects the code not in a function in lecture_5.py

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

Announcements

Think about this for one moment

We can import our own files and our own functions – cool

But if we have any Test-Code in the python file, that code will be run every time we use import

Test-Code – is function calls, variable definitions, print statements – any code not in a function…

So what can we do to fix this?

Import

Let’s copy the lecture_5.py code to a new file “lecture_5_v2.py"

Import

Let’s copy the lecture_5.py code to a new file “lecture_5_v2.py"

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))
```

Import

Let's copy the lecture_5.py code to a new file "lecture_5_v2.py"

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))
```

And add a new function declaration called main()

Import

Let's copy the lecture_5.py code to a new file "lecture_5_v2.py"

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))
```

And let's move the code not in a function in to main()

Import

If we import the new lecture_5_v2 into lecture_6.py

Import

If we import the new lecture_5_v2 into lecture_6.py

```
from lecture_5_v2 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  10
# number_2 is  14
# 3.0
```

Import

If we import the new lecture_5_v2 into lecture_6.py
and call “lecture_6.py” from the command line and view the output

We can see we are now getting the correct output

```
from lecture_5_v2 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  10
# number_2 is  14
# 3.0
```

Import

If we import the new lecture_5_v2 into lecture_6.py
and call “lecture_6.py” from the command line and view the output

We can see we are now getting the correct output

```
from lecture_5_v2 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  10
# number_2 is  14
# 3.0
```

Everything is good in the world, and I’m happy ☺

Import

If we import the new lecture_5_v2 into lecture_6.py
and call “lecture_6.py” from the command line and view the output

We can see we are now getting the correct output

```
from lecture_5_v2 import average_of_two
number_one = 10
number_two = 14
print(average_of_two(number_one, number_two))

# output
# number_1 is  10
# number_2 is  14
# 3.0
```

Everything is good in the world, and I’m happy ☺
But let’s just check that we get the same output from lecture_5_v2
like we previously got from lecture_5

Import

If we now call “lecture_5_v2.py” from the command line and view the output

Import

If we now call “lecture_5_v2.py” from the command line and view the output

We are expecting

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

Import

If we now call “lecture_5_v2.py” from the command line and view the output

We are expecting

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

We can see there is no output

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_5_v2.py
Jasons-MacBook-Pro:code_snippets jasonquinlan$
```

Import

If we now call “lecture_5_v2.py” from the command line and view the output

We are expecting

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

We can see there is no output

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_5_v2.py
Jasons-MacBook-Pro:code_snippets jasonquinlan$
```

Look’s like I broke it again ☹

Import

The reason being there are now 2 functions and no test-code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))
```

Import

The reason being there are now 2 functions and no test-code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))
```

Import

The reason being there are now 2 functions and no test-code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))
```

And nothing will run unless I call a function

Import

So, let's add one more piece of code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))


if __name__ == "__main__":
    main()
```

Import

So, let's add one more piece of code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))


if __name__ == "__main__":
    main()
```

If we ignore the line starting with if… (we will come back to it)

Import

So, let's add one more piece of code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))


if __name__ == "__main__":
    main()
```

If we ignore the line starting with if… (we will come back to it)
We see that we have a call to main()

Import

So, let's add one more piece of code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))


if __name__ == "__main__":
    main()
```

And this call to main() will call the function definition main()

Import

So, let's add one more piece of code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))


if __name__ == "__main__":
    main()
```

And this call to main() will call the function definition main()
And this function will call our code

Import

So, let's add one more piece of code

lecture_5_v2.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    average = (number_1+number_2)/2
    # return the average number
    return average


def main():
    number_one = 2
    number_two = 4
    print(average_of_two(number_one, number_two))


if __name__ == "__main__":
    main()
```

And this call to main() will call the function definition main()
And this function will call our code - Let's just check the output

Import

If we call “lecture_5_v2.py” from the command line and view the output

We can see we are now getting the correct output

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_5_v2.py
number_1 is  2
number_2 is  4
3.0
```

Import

If we call “lecture_5_v2.py” from the command line and view the output

We can see we are now getting the correct output

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_5_v2.py
number_1 is  2
number_2 is  4
3.0
```

And if we call lecture_6, does this still work?

Import

If we call “lecture_5_v2.py” from the command line and view the output

We can see we are now getting the correct output

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_5_v2.py
number_1 is  2
number_2 is  4
3.0
```

And if we call lecture_6, does this still work?

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_6.py
number_1 is  10
number_2 is  14
12.0
```

Yes it does, all is good again ☺

Import

If we call “lecture_5_v2.py” from the command line and view the output

We can see we are now getting the correct output

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_5_v2.py
number_1 is  2
number_2 is  4
3.0
```

And if we call lecture_6, does this still work?

```
Jasons-MacBook-Pro:code_snippets jasonquinlan$ python3 ./lecture_6.py
number_1 is  10
number_2 is  14
12.0
```

Yes it does, all is good again ☺

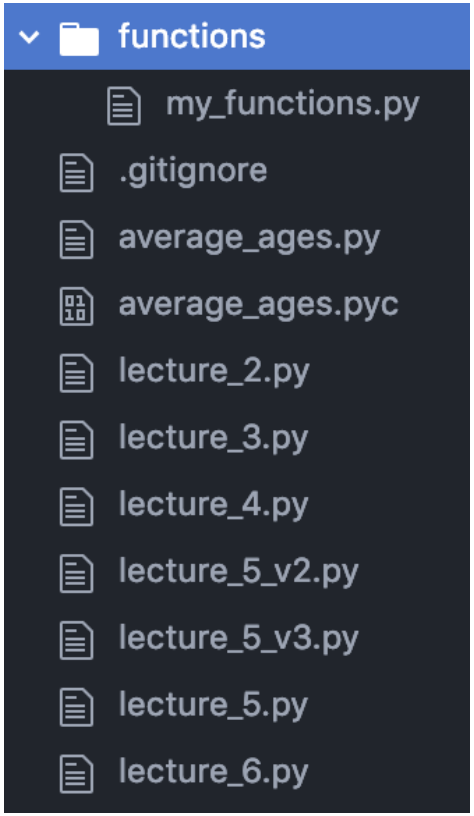
Now before I break it again, let’s look at one more import option

Import

This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder

Import

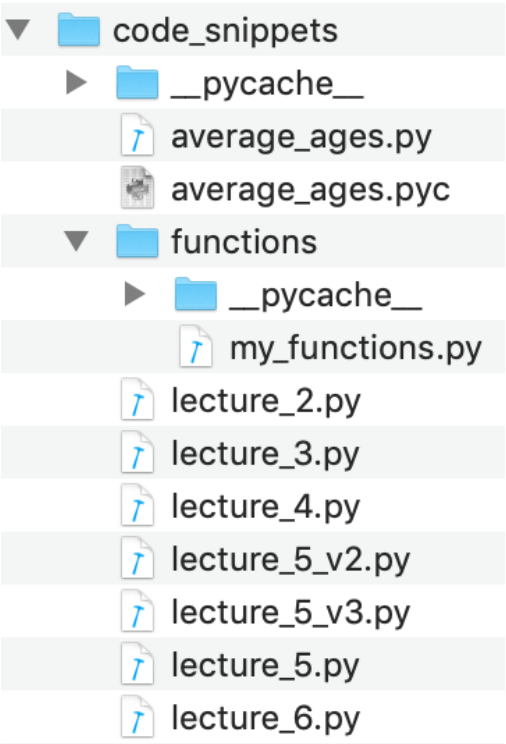
This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder



So let's create a folder "functions" and
a new file in that folder called "my_functions.py"

Import

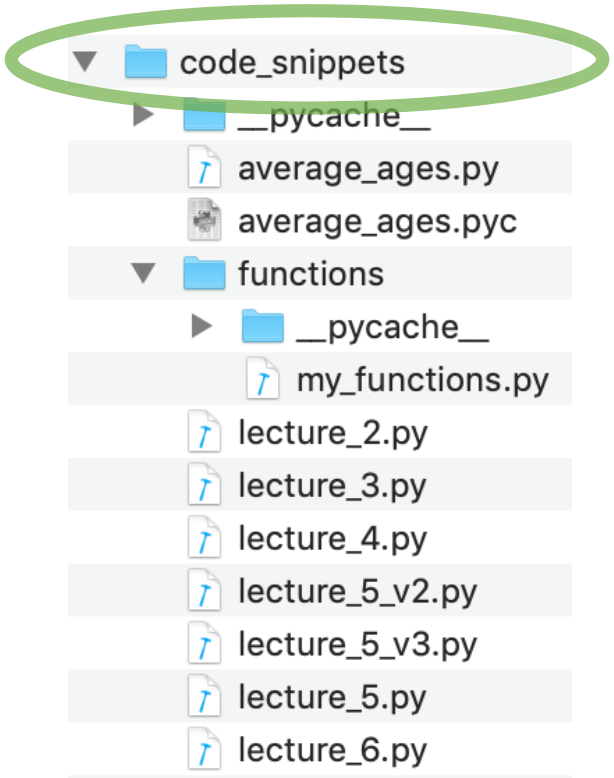
This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder



Let's look at this directory structure from "mac folder view"

Import

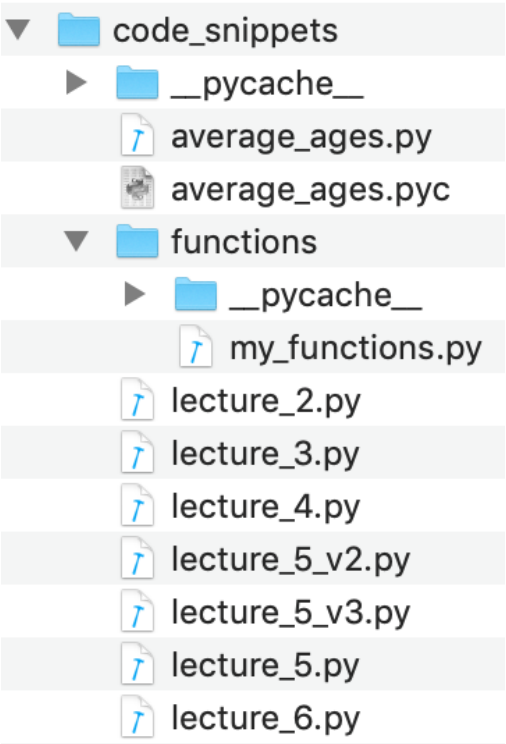
This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder



You can see an outer most folder called (code_snippets)
The outer most folder can also be known as the parent folder

Import

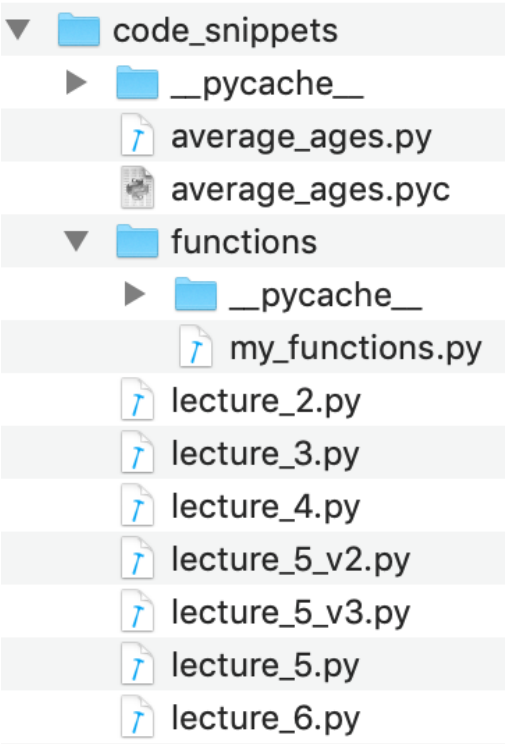
This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder



And within the code_snippets folder we have a number of files (lectures_2.py to lectures_6.py) and a functions folder

Import

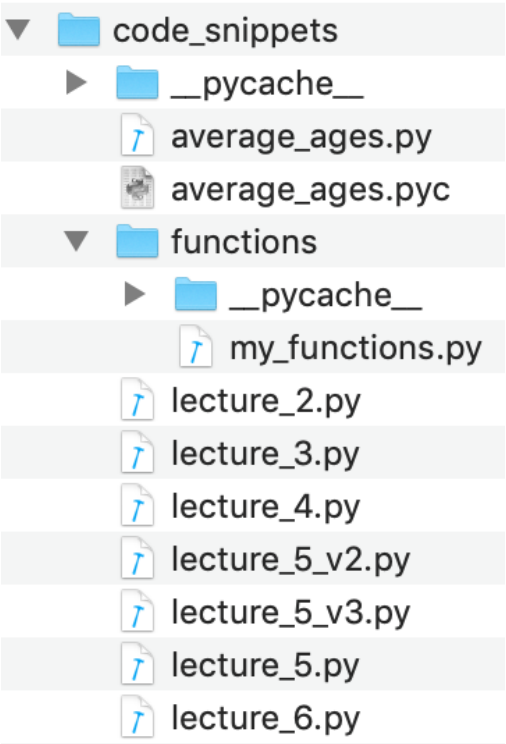
This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder



And finally within the functions folder, we have a file called "my_functions.py"

Import

This is all fine if the file you want to import is in the same folder as your calling file, but what if the function you want is in a file in a nested folder



The functions folder, is known as a nested folder below the code_snippets folder

Import

Let's copy our "average_of_two" function into the new "my_functions.py" file

my_functions.py | lecture_6.py

```
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    # average = (number_1+number_2)/2
    # return the average number
    return "hello"
```

Import

Let's copy our "average_of_two" function into the new "my_functions.py" file

my_functions.py | lecture_6.py

```
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    # average = (number_1+number_2)/2
    # return the average number
    return "hello"
```

I've changed the return statement, so we will see some new output

Import

And let’s see how we import my_functions.py into lecture6.py

my_functions.py | lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

Import

And let’s see how we import my_functions.py into lecture6.py

my_functions.py lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

We call functions.my_functions

Import

And let's see how we import my_functions.py into lecture6.py

my_functions.py | lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

We call functions.my_functions

Note: the use of the dot operator (to denote folder structure)

Import

And let's see how we import my_functions.py into lecture6.py

my_functions.py | lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

The dot operator states that "my_functions" is a file within the folder "functions" and that is where we should look

Import

And let's see how we import my_functions.py into lecture6.py

my_functions.py | lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

The dot operator states that "my_functions" is a file
within the folder "functions" and that is where we should look
So the dot operator is also mutable
(allowing us to import files and call functions) - "string".format()

Import

And let's see how we import my_functions.py into lecture6.py

my_functions.py | lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

Finally and most importantly, our output looks good ☺

Import

And let's see how we import my_functions.py

my_functions.py | lecture_6.py

```
from functions.my_functions import average_of_two

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

# output
# number_1 is  2
# number_2 is  4
# hello
```

Finally and most importantly, our output looks good ☺

Note: the "hello"

Import

my_functions.py | lecture_6.py

```python
def average_of_two(number_1, number_2):
    """ this is my 'docstring'
    function to determine the average of two numbers
    number_1: first int to pass in
    number_2: second number to pass in
    """
    print("number_1 is ", number_1)
    print("number_2 is ", number_2)
    # determine the average of two numbers
    # average = (number_1+number_2)/2
    # return the average number
    return "hello"
```

import Recap

- So a very quick look back at import:
- We now know how to import functions from Python libraries
- We now know how to import functions we create
  - From both files within the same folder as the calling file
  - And from folders nested below the calling file:
    - functions/my_functions.py
- For now, we will not be looking at calling functions in outer/parent folders

Live Coding Time…

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh