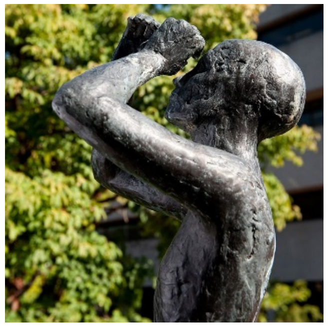


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Semester 1 revision

If any of the content, we cover in these revision lectures is confusing, ask questions

If not in class, ask on the anonymous google form

We will then cover the content in the next class or in the extra coding class

This is your chance to get to know this material

☺

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Semester 1 revision

Variables:

Any lowercase word

Should be descriptive of the value it will hold

Semester 1 revision

Variables:

Any lowercase word

Should be descriptive of the value it will hold

Do not use keywords, do not use shadowing, do not:

1. have a number as first character – 7up
2. have a "#" in the variable name – mixture#3

Python reserved keywords

Just known that within Python

these 33 reserved keywords exist

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Semester 1 revision

Shadowing:

Using Python library functions as variable names

```
list =  [1,2,3]
my_hello_list = list(“hello”)
```

```
TypeError: 'list' object is not callable
```

Rather than use Pythons list() function, to create a list from “hello”, Python tries to use my variable “list”

Data Types

If we write “print(type(x))” for each example, we get the type:

Type	Example	type()

Numeric: Integer, Float	x = 10
	x = 10.0
		<class 'int’>
		<class 'float'>

String	x = “Mike”	<class 'str’>
Boolean	x = True, x = False	<class 'bool'>
List	x = [10, 20, 30]	<class 'list'>
Tuple	x = ("Ed", "Edd", "Eddy", 2009)	<class 'tuple'>
Dictionary	x = {‘one’: 1, ‘two’: 2}	<class 'dict'>
List	x = ["Ed", "Edd", "Eddy", 2009]	<class 'list'>

Semester 1 revision

Functions:

Similar to variables,
as it’s a means of reducing the amount of code you write

But, typically not for single lines of code,
but for a number of lines of code (statement block)

Semester 1 revision

- Let’s define a function called “ask_for_int_input”

def – keyword for Python function

Function name – snake case ask_for_int_input

Parameter(s) passed to the function – zero or more question_string

```python
def ask_for_int_input(question_string):
```

Function definition ends in “colon” :

Semester 1 revision

- So a question to ask yourself is:
- Do we need to return a value from our function?

- In this instance, the answer is yes, as we need the input from the user (keyboard)

- So, let’s add the return statement and some comments:

```
‘’’ a function that returns an integer value from the user ‘’’
def ask_for_int_input(question_string):
    # save the input from the user in a variable called "user_input"
    user_input = int(input(question_string))
    # return the content of the variable called "user_input"
    return user_input
```

Python Functions

So we've created a function - how do we call it?

```python
def ask_for_int_input(question_string):
```

```python
age1 = ask_for_int_input("Please enter age 1: ")
```

Returned value

Function call
ask_for_int_input()

Function parameter

Python Functions Recap

- 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

Semester 1 revision

return:

```
def hello_you():
    return

print(type(hello_you()))
```

None - If the function has no return statement

Or

None - If the function has an empty return statement

```
<class 'NoneType'>
```

Semester 1 revision

return:

```
def hello_you():
    return 7

print(type(hello_you()))
```

Type of value - If a single value is returned

```
<class int'>
```

Semester 1 revision

return:

```
def hello_you():
    return 7, 8

print(type(hello_you()))
```

Tuple of values - If two or more values are returned

```
<class ‘tuple'>
```

Python Operators

We have see:

assign (=), addition (+) and float division (/)

Other Python operators include:

multiply (*), subtract (-), exponent (**)

modulus (%), integer division (//)

Python Operators

Note: Operators are mutable – they change their operation depending on the data type they're working with.

We have seen this with (+)

String concatenation "Hello " + "World" –> "Hello World"

Addition 2 + 3 -> 5

Python Operators

Another good example of this mutability is with multiply (*):

3 * 5 -> 15

print(”Beetlejuice ” * 3) -> "Beetlejuice Beetlejuice Beetlejuice "

Python Operators

- Float division: / produces a float (13/6 produces 2.1666)
- Integer division: // produces an integer (13//6 produces 2)
- You may need to cast a value to change the data type to get the desired output.

```
x = 13
y = 6
z = x // y
print (z)            #Output is 2
z = float(z)
print (z)            #Output is 2.0
```

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Python Operators

The % operator is called the *modulus*.

It will tell you the **remainder** after integer division.

```
x = 11 % 3              #output is 2

x = 10 % 2              #output is 0
```

We can use // and % to calculate the answer and remainder:

```
print(10 // 3)          #output is 3

print(10 % 3)           #output is 1
```

The exponent (power of) operator is **

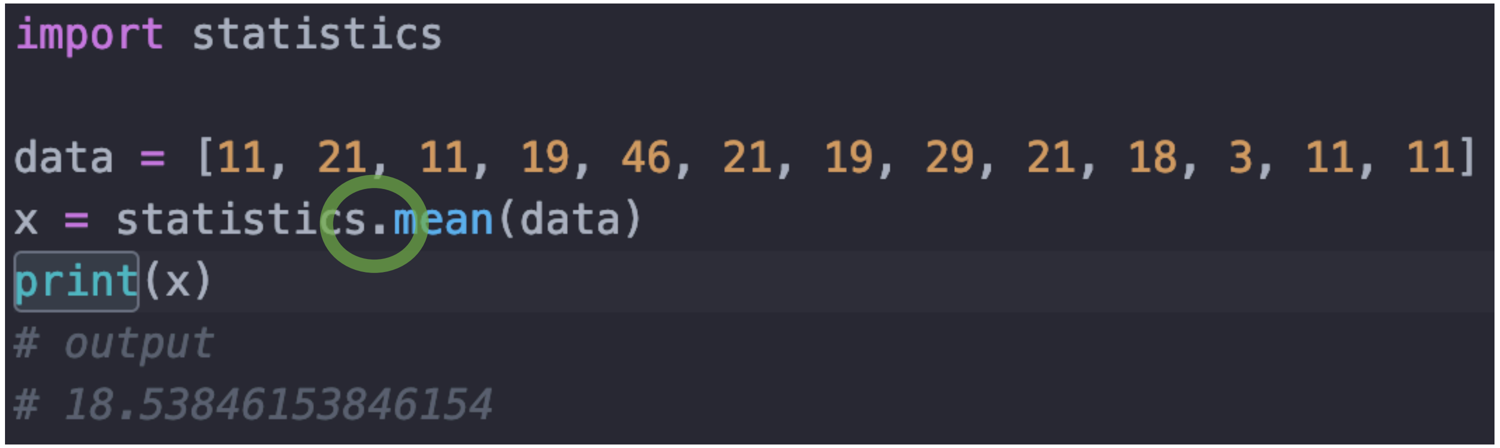
```
x = 2 ** 3              #output is 8
```

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
```

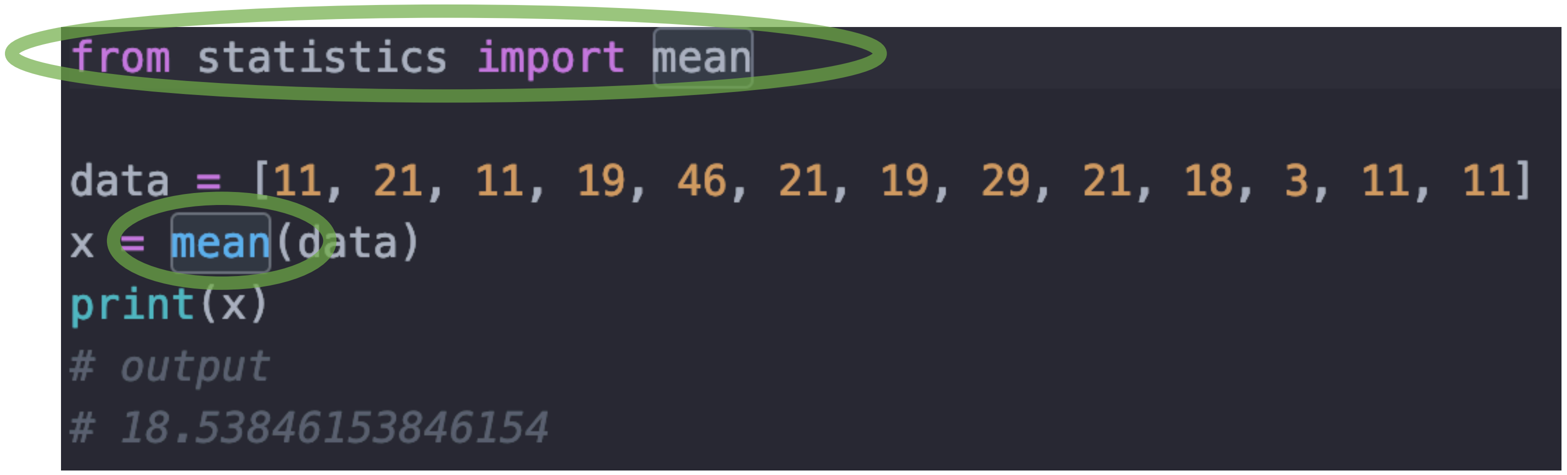
Mean is called using the dot (.) operator

Using Python Functions

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

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

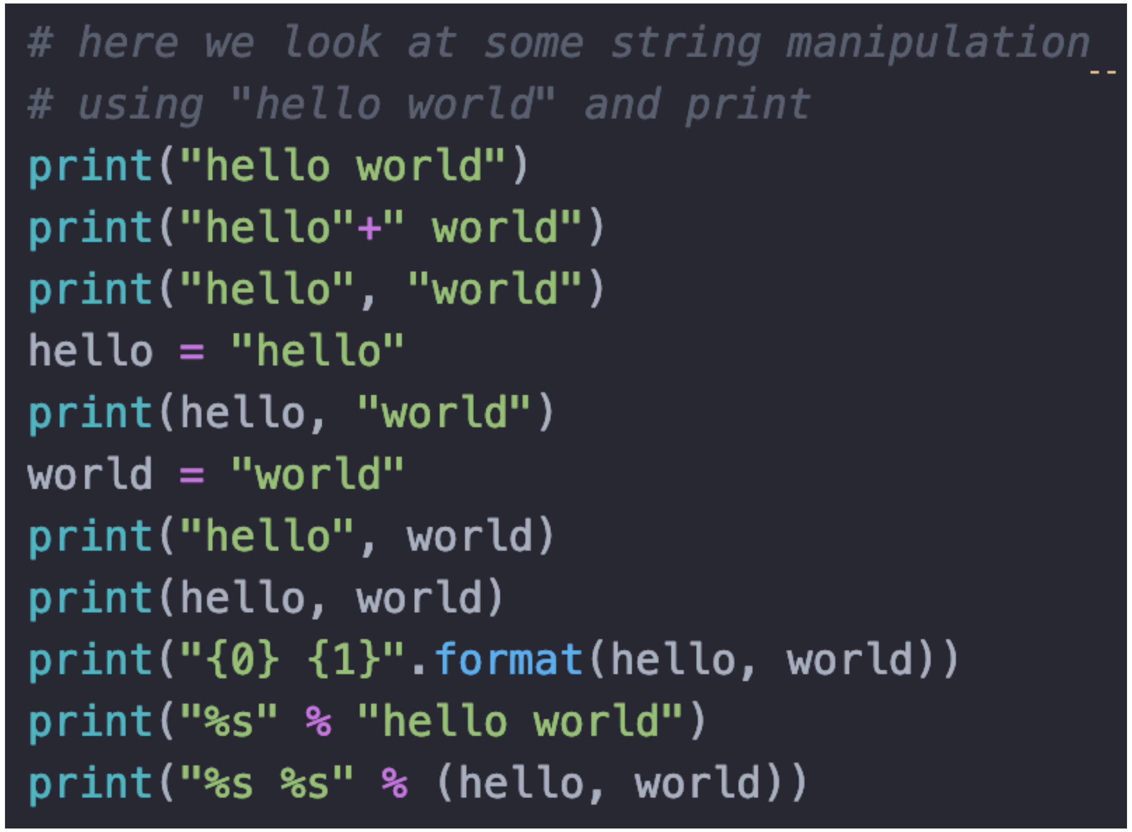
If we only want to use mean(), we can modify our import



```
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
```

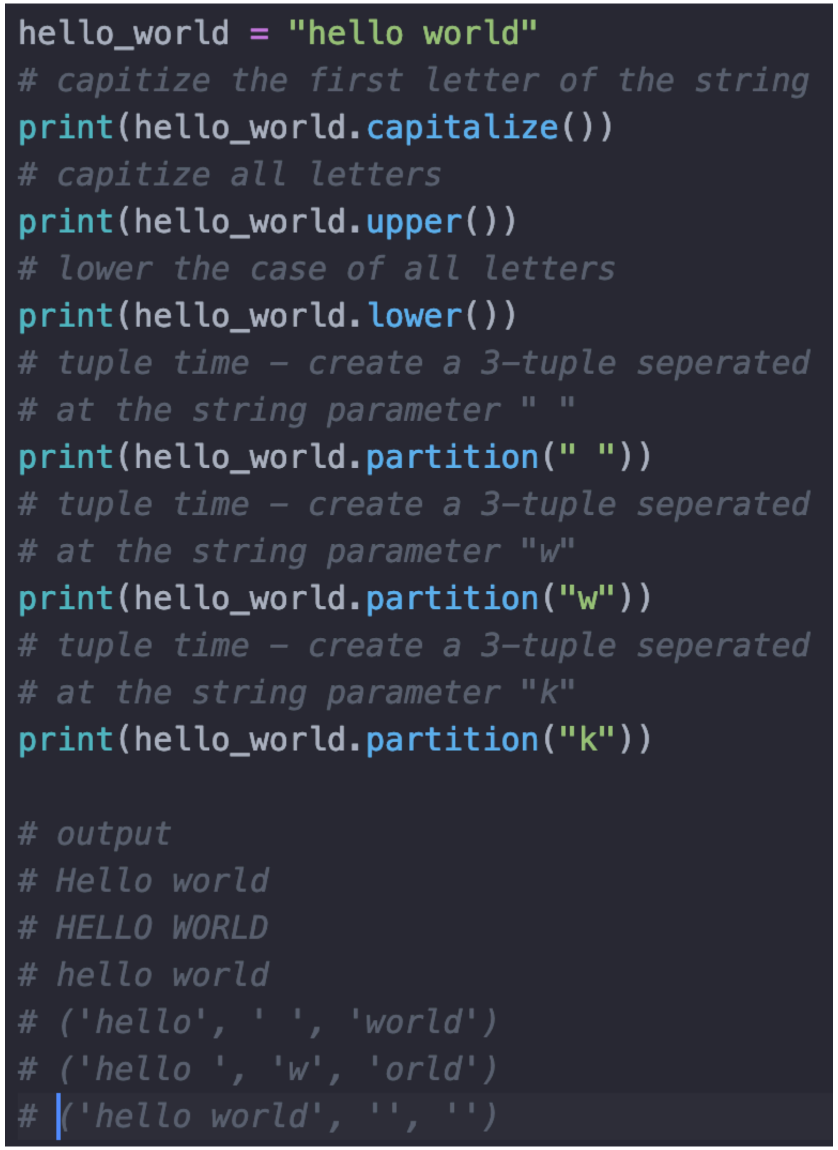
String Manipulation



```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))
```

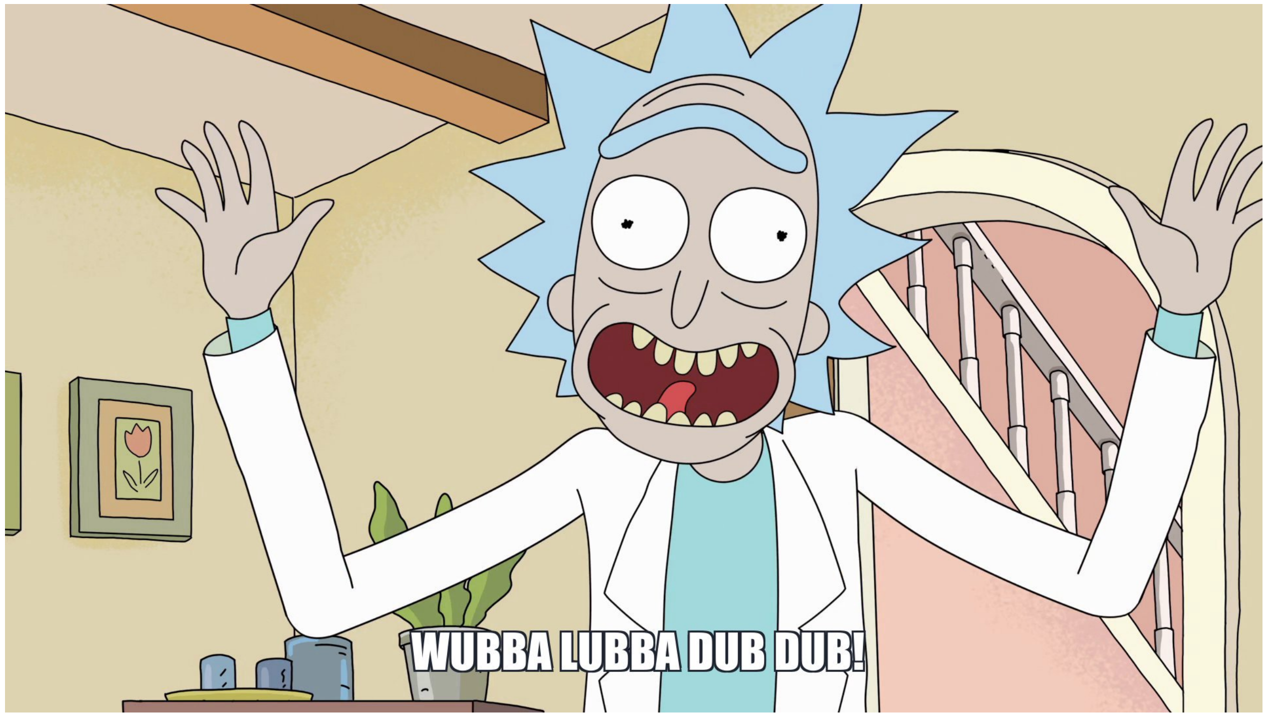
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

String 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', '', '')
```


WUBBA LUBBA DUB DUB!