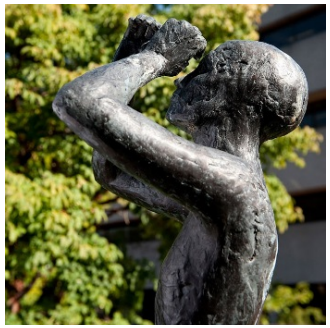


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland

Coláiste na hOllscoile Corcaigh

Continuous Assessment Lab

Continuous Assessment Lab

The CA lab will be released next Monday 11th November

Deadline for submission is the 23rd November @ 1am

The lab will be worth 15 marks (5%) of your total marks

The lab will cover CS1117 weeks 1 to 9 (inclusive)

I will be using “turnitin” – a plagiarism program in Canvas – to check for repeating code between submissions

Continuous Assessment Lab

The Tuesday/Wednesday labs will be open

But the demonstrators and I will not be able to answer any CA coding question you have

We can only clarify the questions I ask in the CA assignment

i.e., “What do I mean by question X, what is expected as a returned value, etc…”

But we can answer any question you have from Labs 1 to 8

Continuous Assessment Lab

In the CA, similar to the other labs

I will tell you what the functions are called, what the parameters are, and what is expected to be returned.

So make sure you name the functions exactly, add the parameters exactly and return exactly what is asked for…

I may give you some examples of function calls

and expected output

But expect me to test with a lot more function calls, so make sure you test with as many calls as you can think of.

Continuous Assessment Lab

I have not decide exactly

how to evaluate the lab submissions:

I will either:

1. Call the functions 15 to 30 times and allocate 1 mark or .5 marks (respectively) for each correct return value

2. Or I may review each line of code and allocate marks based on define function call, loop definitions, docString, variable/function names, comments, etc.

3. Or a combination of both…

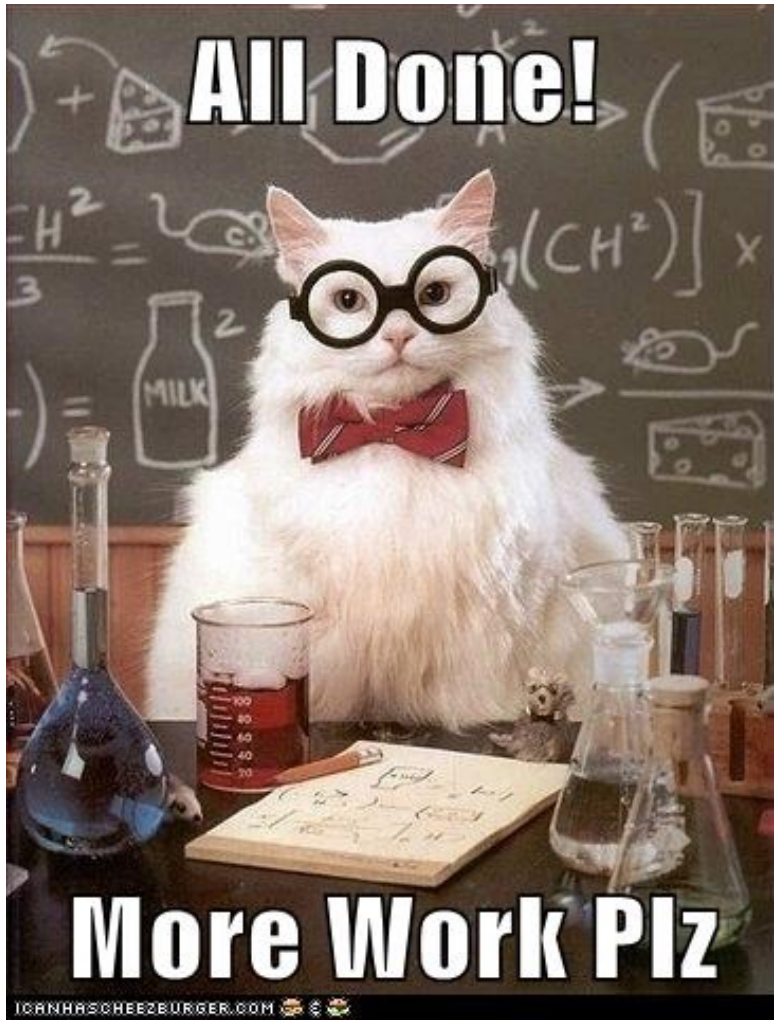
Continuous Assessment Lab

Best of luck with it ☺

Canvas Student App

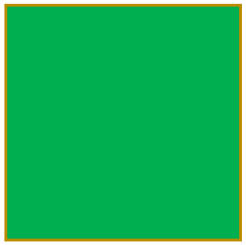
Let's Sign into this lecture now

Access Code
35234



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Dictionaries

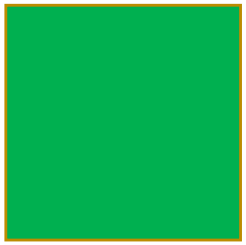


Before we start on Dictionaries, let's look back at Tuples and Lists:

Both are an ordered collection of elements, i.e.

The relative location of the element is determined by the order in which you added the element

Dictionaries



Before we start on Dictionaries, let's look back at Tuples and Lists:

```
my_list = [4, 5, 6, 7]
print(my_list[2])  # returns 6
```

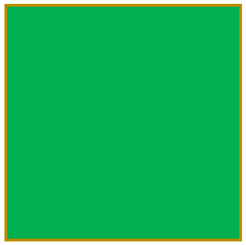
When I print my_list[2], I get back 6

As 6 was the third element I added

And I added it to index 2

Same for tuples

Dictionaries



Dictionary defined using curly brackets:

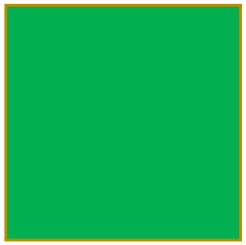
```
my_dictionary = {} # empty dictionary
```

```
my_dictionary = {key:value}
```

key is a unique item

value can be a recurring item

Dictionaries



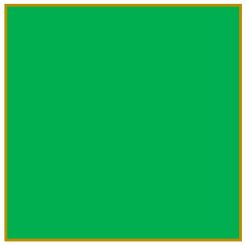
Dictionary defined using curly brackets:

my_dictionary = {} # empty dictionary

my_dictionary = {key:value}

```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

Dictionaries



Once we have our dictionary defined we can:

```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

Get the keys, using my_dictionary.keys()

```
print(my_dictionary.keys())
# dict_keys(['February', 'March', 'April'])
```

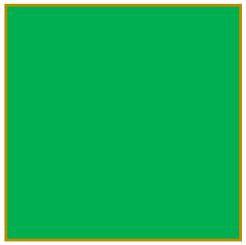
And get values, using my_dictionary.values()

```
print(my_dictionary.values())
# dict_values(['Spring', 'Spring', 'Spring'])
```

Or everything using my_dictionary.items()

```
print(my_dictionary.items())
# dict_items([('February', 'Spring'), ('March', 'Spring'), ('April', 'Spring')])
```

Dictionaries



You will notice from these calls, that the keys and names are coming back in the order we added them:

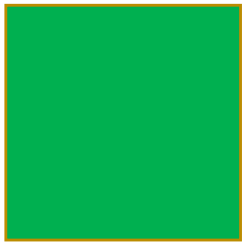
```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

```
print(my_dictionary.keys())
# dict_keys(['February', 'March', 'April'])
```

```
print(my_dictionary.values())
# dict_values(['Spring', 'Spring', 'Spring'])
```

```
print(my_dictionary.items())
# dict_items([('February', 'Spring'), ('March', 'Spring'), ('April', 'Spring')])
```

Dictionaries



Because of these functions we can use:

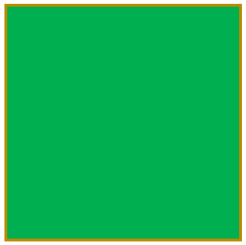
```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

```
if "February" in my_dictionary.keys():
    print(True)
# True
```

```
if "Spring" in my_dictionary.values():
    print(True)
# True
```

```
for key, value in my_dictionary.items():
    print(key, value)
# February Spring
# March Spring
# April Spring
```

Dictionaries



```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

we can get values via the key

```
print(my_dictionary["February"])
# Spring
```

We can also re-assign values to keys

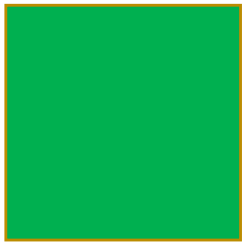
```
my_dictionary["February"] = "Swinter"
print(my_dictionary["February"])
# Swinter
```

Note how we assign via a key and not an index

This makes dictionaries “an un-ordered structure”

```
print(my_dictionary.items())
# dict_items([('February', 'Swinter'), ('March', 'Spring'), ('April', 'Spring')])
```

Dictionaries



```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

If we try and get values for a key that does not exist, we get an exception:

```
print(my_dictionary["August"])
# KeyError: 'August'
```

When we want to add a key/value to the dictionary, we just using assign '='

```
my_dictionary["August"] = "Swinter"
print(my_dictionary["August"])
# Swinter
```

There is no need for append() here

List

In List we can use all the basic functions for Tuple:

We can get a value based on index number – <list>[index]

We can get the length of a List – len(<list>)

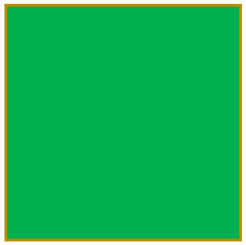
We can add to a List – mutable – <list>[4] = value

We can count how many times a value appears in a List – <list>.count(value)

And we can get the index based on value - <list>.index(value) (throws exception) if the value does not exist in the List

Oh and we can use - if value in <list>:

Dictionaries - Recap



From the basic functions:

We can get a value based on key – < dict >[key]

We can get the length of a Dict – len(< dict >)

We can add to a Dict – mutable – < dict >[key] = value

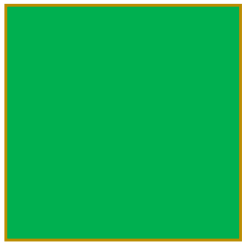
We can not use count as the keys are unique

And we can not use index as we have only keys

Oh and we can use – if/for item in <list>.keys()/values():

for key, value in <list>. Items():

Dictionaries - Recap



From the basic functions:

We can get a value based on key – < dict >[key]

We can get the length of a Dict – len(< dict >)

We can add to a Dict – mutable – < dict >[key] = value

We can not use count as the keys are unique

And we can not use index as we have only keys

Oh and we can use – if/for item in <list>.keys()/values():

for key, value in <list>. Items():

Question: how could we use count() on the values???

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh