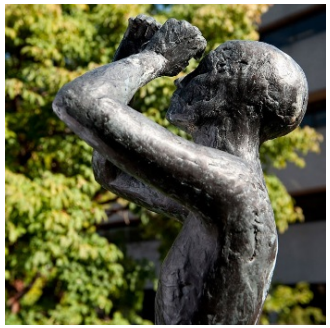


CS1117 – Introduction to Programming

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

Continuous Assessment 3

This Multiple Choice Quiz covers weeks 6 to 11 inclusive

This is a good chance for you to see if you understand what we have covered in these weeks

Scope and List Comprehensions will be on MCQ-2

As we have not completed a Lab in these concepts, I advise undertaking some coding with them

Announcements

Continuous Assessment 3

Available only on Canvas

So you will need a laptop, tablet, etc, to take the quiz.

If you do not have one of these, please let me know by email and I will arrange alternative access for the quiz.

If you do not received an email from me allocating you a space with alternative access, you must be in this room to access the MCQ.

Announcements

Continuous Assessment 3

Available only on Canvas

You will need access to Eduroam WiFi
so make sure you have signed up

IP filtering will be used for access to the quiz

Announcements

Continuous Assessment 3

Available only on Canvas

A code will be needed to access the quiz

This will be given out at the beginning of the class

Announcements

Continuous Assessment 3

Mobile phones will be turned off and placed on the desk in front of you.

You should not access online website for answers during the quiz

If you are seen surfing these sites, you will get a zero grade for this quiz.

Announcements

Continuous Assessment 3

This work must be your own,
so no asking your neighbour for answers

Do not take answers from others machines

There is no guarantee they are correct :)

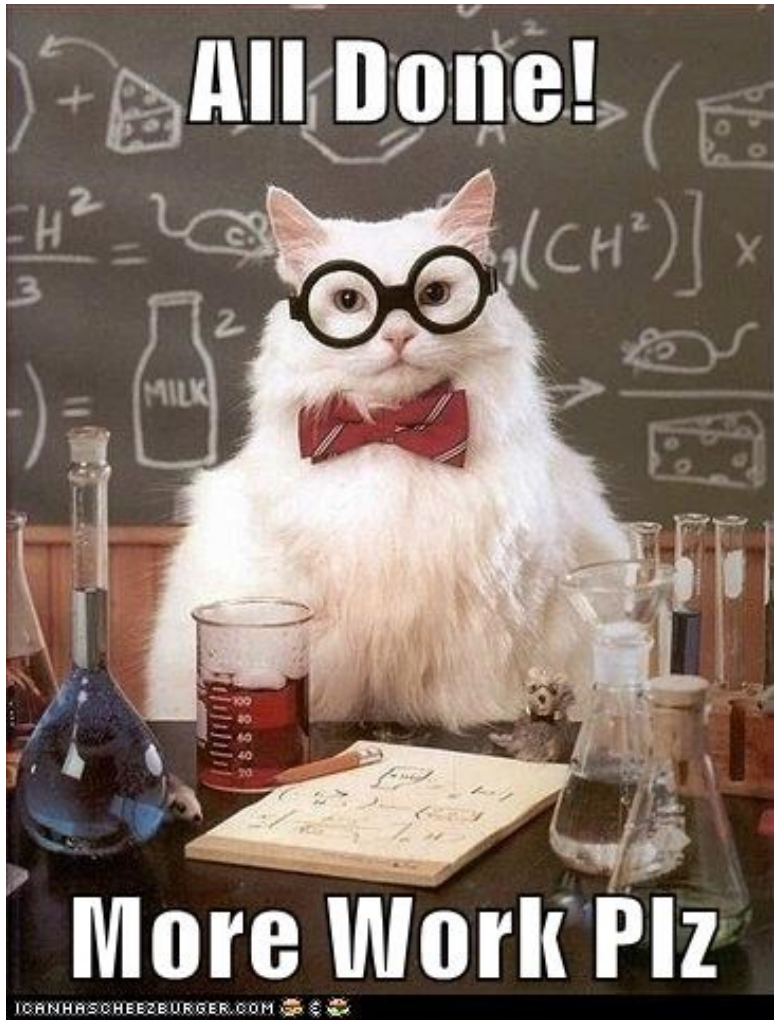
You can bring pen and paper for rough work

Rough work does not need to be handed up.

Canvas Student App

Let’s Sign into this lecture now

Access Code
89076



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Continuous Assessment Lab

Continuous Assessment Lab

The CA lab was released last Monday morning

Deadline for submission is this Saturday 23rd November @ 1am

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

This lab is part of 6 labs with 5 of your maximum scores being used for your CA mark

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 questions 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 will give you six examples for each function call

and associated 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

For the grading, I will be:

1. Calling each of the functions 10 additional times and allocating half a mark for each correct return value. I will not be grading the examples I give you.

2. My tests assume a returned value from your functions, so do not use print. Print will get you no marks!!!!

3. I will also review each line of your code and if you use Python library functions that you were told not to use, I will deduct all marks for that function. Only use functions we have covered in class…

4. If you do not follow all steps in the assignment, I will deduct marks.

Continuous Assessment Lab

Very, very important:

Make sure you submit on time

Double check to make sure you have submitted

I will not be accepting any of the CA submissions via email

Continuous Assessment Lab

Very, very important:

Make sure you test your code…

Test you code with my 6 examples

Then think of as many other ways to test it, and retest

Test your code from main.py and not functions.py

When you upload your functions.py file, re-download it and retest it.

Continuous Assessment Lab

To clarify:

```
print(to_english(142))
# "One hundred and forty two" - wrong
# One hundred and forty two - correct
```

I do not want *to_English(n)* to return with quotes

No need to add \"<text>\" to your return string

I state in the exercise sheet:

return the string "One hundred and forty two"

(excluding the quotes)

Continuous Assessment Lab

To clarify:

When you must return multiple values

It is easier to create variables

val_1 = <result_1>
val_2 = <result_2>
return val_1, val_2

Do not print

Continuous Assessment Lab

To clarify:

Do not use List Comprehensions in your functions

This CA assessment lab covers weeks 1 to 9

We are covering List Comprehensions
in week 10 & 11

Continuous Assessment Lab

Best of luck ☺

Continuous Assessment Lab

I've been asked, to ask you to fill out a

Computer Science Feedback Form

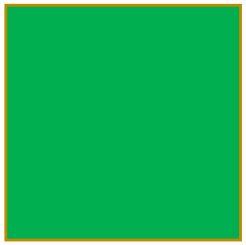
So, go to Canvas, go to Feedback and you will see a link for "Computer Science Feedback Form"

12 questions, so please take 15 minutes to fill it out now

BSc CS students only…

List Comprehension

List Comprehension



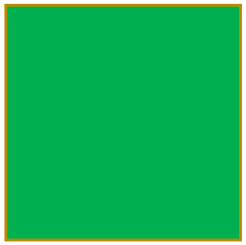
let’s look back at return_list(number):

```
def return_list(number):
    # return a list from 0 to 9
    x = []
    for i in range(number):
        x.append(i)
    return x
```

We can rewrite this as a list comprehension:

```
def return_list(number):
    # return a list from 0 to 9
    return [i for i in range(number)]
```

List Comprehension



```
def return_list(number):
    # return a list from 0 to 9
    return [i for i in range(number)]
```

List comprehension is short hand for creating lists

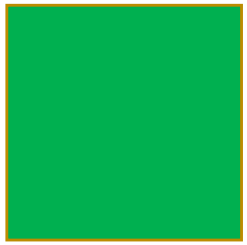
Using the syntax:

result = [*expression* *iteration* *filter*]

Create a list of even numbers:

x = [i for i in range(number) if i % 2 == 0]

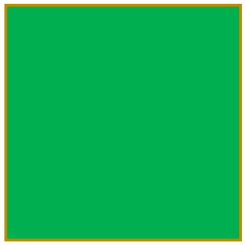
List Comprehension



Let’s look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

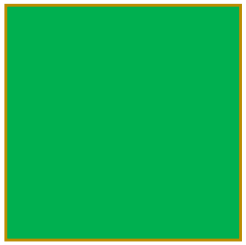


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

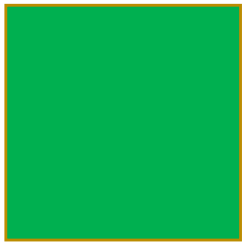


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

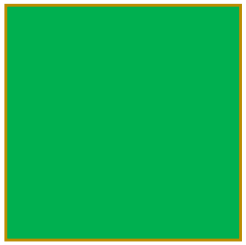


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

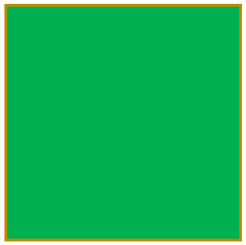


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

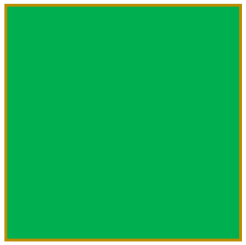


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

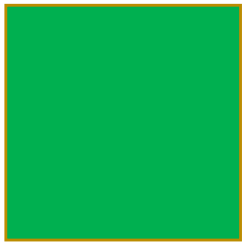


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension

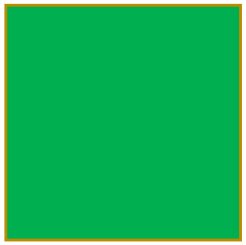


Let's look at this example:

```
def return_even_list(number):
    x = []
    for i in range(number):
        if i % 2 == 0:
            x.append(i)

    return x


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

```
def return_even_list(number):
    return [i for i in range(number) if i % 2 == 0]


print(return_even_list(10))
# 0, 2, 4, 6, 8]
```

List Comprehension



Every list comprehension can be rewritten as a `for` loop but not every `for` loop can be rewritten as a list comprehension.

The key to understanding when to use list comprehensions is to practice identifying problems that *smell* like list comprehensions.

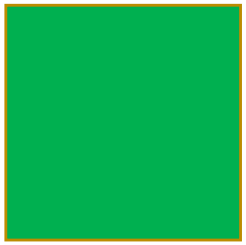
If you can rewrite your code to look *just like this* `for` *loop*, you can also rewrite it as a list comprehension:

```
new_things = []
for ITEM in old_things:
    if condition_based_on(ITEM):
        new_things.append("something with " + ITEM)
```

You can rewrite the above `for` loop as a list comprehension like this:

```
new_things = ["something with " + ITEM for ITEM in old_things if condition_based_on(ITEM)]
```

List Comprehension



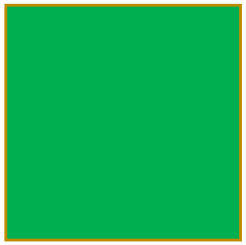
```
doubled_odds = []
for n in numbers:
    if n % 2 == 1:
        doubled_odds.append(n * 2)
```

```
doubled_odds = [n * 2 for n in numbers if n % 2 == 1]
```

We copy-paste from a `for` loop into a list comprehension by:

1. Copying the variable assignment for our new empty list
2. Copying the expression that we've been `append`-ing into this new list
3. Copying the `for` loop line, excluding the final `:`
4. Copying the `if` statement line, also without the `:`

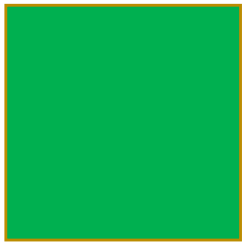
List Comprehension



Examples

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)

squares = [x**2 for x in range(10)]
print(squares)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

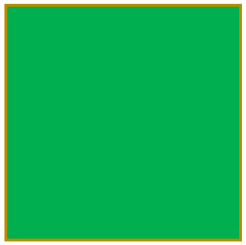
List Comprehension



Examples

```
lower = [x.lower() for x in ["A", "B", "C"]]
print(lower)
# ['a', 'b', 'c']
```

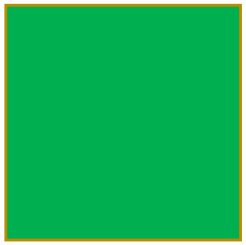
List Comprehension



Examples

```
upper = [x.upper() for x in ["a", "b", "c"]]
print(upper)
# ['A', 'B', 'C']
```

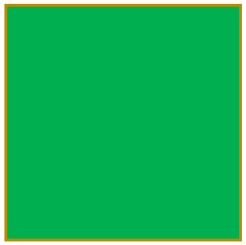
List Comprehension



Examples

```
def double(x):
    return x*2


doubler = [double(x) for x in range(10)]
print(doubler)
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

List Comprehension



Examples

```
multi_check = []
for x in [1, 2, 3]:
    for y in [10, 20, 30]:
        multi_check.append(x+y)
print(multi_check)

multi_check = [x+y for x in [1, 2, 3] for y in [10, 20, 30]]
print(multi_check)
# [11, 21, 31, 12, 22, 32, 13, 23, 33]
```

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh