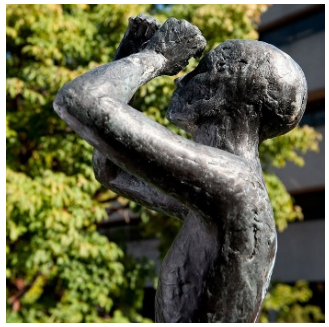


CS1117 – Introduction to Programming

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

☺

Semester 1 revision

Week 5

Lecture 14 - continued

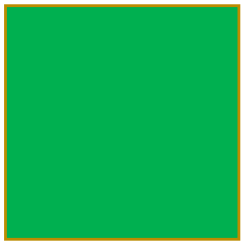
While

Your Turn – Calculate the answer

```
def puzzle(n):
    v1 = 2
    v2 = 1
    while n > 0:
        if (n % 2 == 1):
            v2 = v1 * v2
        v1 = v1 * v1
        n = n // 2
    return v2


print(puzzle(10))
```

While



Your Turn – Calculate the answer

```
def puzzle(n):
    v1 = 2
    v2 = 1
    print("n = %d, v1 = %d, v2 = %d" % (n, v1, v2))
    while n > 0:
        if (n % 2 == 1):
            v2 = v1 * v2
        v1 = v1 * v1
        n = n // 2
        print("n = %d, v1 = %d, v2 = %d" % (n, v1, v2))
    return v2


print(puzzle(10))
```

While

```
n = 10, v1 = 2, v2 = 1
n = 5, v1 = 4, v2 = 1
n = 2, v1 = 16, v2 = 4
n = 1, v1 = 256, v2 = 4
n = 0, v1 = 65536, v2 = 1024
1024
```

Your Turn – Calculate the answer

```
def puzzle(n):
    v1 = 2
    v2 = 1
    print("n = %d, v1 = %d, v2 = %d" % (n, v1, v2))
    while n > 0:
        if (n % 2 == 1):
            v2 = v1 * v2
        v1 = v1 * v1
        n = n // 2
        print("n = %d, v1 = %d, v2 = %d" % (n, v1, v2))
    return v2


print(puzzle(10))
```

While Recap

We saw lists are a powerful dynamic format in Python

But they can be slow to populate line by line

Boolean logic helps us define True or False statements

We can use if conditional statements to control code flow

Creating a loop in the code can help to reduce workload

When we have a lot of repeating code

While Recap

We introduced While loops as a mechanism for repeating code

We use While loop when we don't know how long the loops will last

While the condition remains True we continue to execute the statement block of code

But… we need some mechanism to make the condition false, otherwise we loop forever…

While Recap

Counters are very important to While loops

They not only allow us to loop in the While a certain number of times

But they also allow us to use list indexing to gather and set information

Incorrect counting can cause the majority of the bugs seen in While loops

While

But what happens when we get caught in an infinite loop

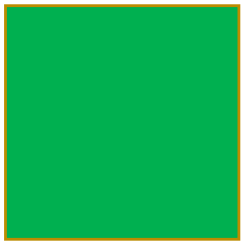
What can we do to get out of the loop

And is there anything we can do when the loop completes

For this we investigate

continue, break and else (yes else – from if/else)

While



break

Is a Python mechanism for stopping loops

It terminates the execution of the
statement block code in the loop

And Python moves out of the while loop and
to the next location in the code

```
print("Phew. The While has stopped")
```

While

We add a Boolean check

```
import time
i = 0
# time in seconds since January 1, 1970 - when time began
# known as "epoch date" - unix start of time
start_time = time.time()

print("start time:", start_time)
while i < 10:
    print(i)
    # i += 1
    # get the current time
    current_time = time.time()
    print("current time:", current_time)
    # if the current time less start time is greater than 3 seconds
    if current_time - start_time > 3:
        print("forced end time:", time.time())
        print("break")
        # break - stops the while loop and continues to next line of code
        break

print("Phew. The While has stopped")
```

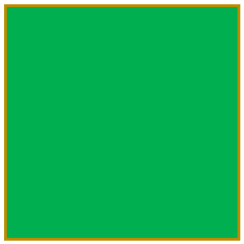
Semester 1 revision

Week 5

Lecture 15

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

While



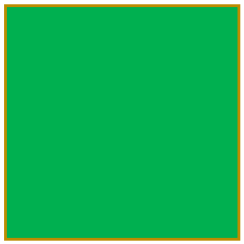
We also have a mechanism for ignoring specific runs through the loops

continue

continue tells Python to ignore the rest of code in the statement block for the current loop

And move to the next loop (iteration)

While



continue example

```
print("Printing odd numbers")
limit = int(input("Provide a maximum number >>> "))

i = 1
while i < limit:
    if i % 2 == 0:
        i += 1
        continue
    print(i, end=" ")
    i += 1

print("\nPhew. The While has stopped")
```

While

Let's look at that "Phew"… one last time

```
print("Printing odd numbers")
limit = int(input("Provide a maximum number >>> "))

i = 1
while i < limit:
    if i % 2 == 0:
        i += 1
        continue
    print(i, end=" ")
    i += 1

print("\nPhew. The While has stopped")

# Output
# Provide a maximum number >> > 10
# 1 3 5 7 9
# Phew. The While has stopped
```

While

As while is so like if, we can actually use else ☺

```
print("Printing odd numbers - v2")
limit = int(input("Provide a maximum number >>> "))

i = 1
while i < limit:
    if i % 2 == 0:
        i += 1
        continue
    print(i, end=" ")
    i += 1
else:
    print("\nPhew. The While has stopped")

# Output
# Provide a maximum number >> > 12
# 1 3 5 7 9 11
# Phew. The While has stopped
```

While

If we can use else, can we use elif ☹

```
print("Printing odd numbers - v3 - elif")
limit = int(input("Provide a maximum number >>> "))

i = 1
while i < limit:
    if i % 2 == 0:
        i += 1
        continue
    print(i, end=" ")
    i += 1
elif i == limit:
    #print("i is equal to limit")
else:
    print("\nPhew. The While has stopped")

# Output
# elif i == limit:
#    ^
# SyntaxError: invalid syntax
```

While

Does break influence else?

```
print("Printing even numbers up to maximum of value of 8 - v2")
limit = int(input("Provide a maximum number >>> "))

i = 1
max_value = 8
while i < limit:
    if i % 2 == 0:
        if i > max_value:
            break
        print(i, end=" ")
    i += 1
else:
    print("\nPhew. The While has stopped")

# Output
# Provide a maximum number >> > 12
# 2 4 6 8
```

Unless while can end properly, the else is never called

While Recap

We saw how to use break, continue and else

To influence the flow of the loops

break – terminates the loop

continue – ignore code within a specific loop and moves to the next loop

else – runs code when the loop ends properly (break can cause else not to be called)

Semester 1 revision

Week 6

Lecture 16

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

While recap

We have a mechanism for looping over repeating code

We use a While loop when we're not sure how many times to execute a piece of code i.e. **indefinite** loop

While the condition remains True,

execute the statement block

Remember we need some way to make the condition False

Otherwise it becomes an infinite loop…

loops

We have a second mechanism for
looping over repeating code

We use a for loop when we know how many times to execute a piece of code i.e. **a count-controlled** loop

for a set number of loops remain True,
and execute the statement block

We do not need a counter to make the condition False

The chances of an infinite loop are reduced

for loops

for loops are normally used when we want to loop over a sequence of data, e.g.

Lists

Tuples

Strings

Dictionaries (to be covered soon)

for loops

As our if and while are both Booleans and we could replace if with while in some cases

Can we replace if and while with for?

Let's see:

for loops

As our if and while are both Booleans and we could replace if with while in some cases

Can we replace if and while with for?

Let's see:

```
val = "e"
item_to_check = "Hello"
if val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

for loops

As our if and while are both Booleans and we could replace if with while in some cases

Can we replace if and while with for?

Let's see:

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

for loops

As our if and while are both Booleans and we could replace if with while in some cases

Can we replace if and while with for?

Let's see:

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

for loops

As our if and while are both Booleans and we could replace if with while in some cases

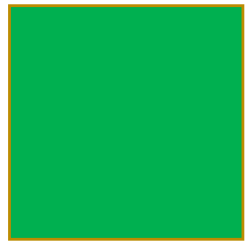
Can we replace if and while with for?

Let's see:

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

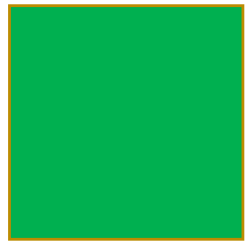
What is the output??

for loops



```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

for loops



```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

```
H is a character in Hello
e is a character in Hello
l is a character in Hello
l is a character in Hello
o is a character in Hello
```

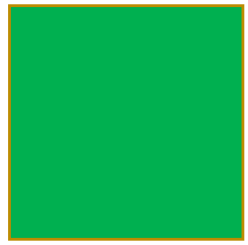
for loops

for every value in item_to_check, print the value

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

```
H is a character in Hello
e is a character in Hello
l is a character in Hello
l is a character in Hello
o is a character in Hello
```

for loops



```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

```
H is a character in Hello
e is a character in Hello
l is a character in Hello
l is a character in Hello
o is a character in Hello
looks like o is not a character in Hello
```

Oops….

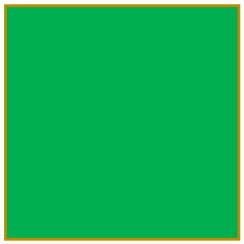
for loops

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

```
H is a character in Hello
e is a character in Hello
l is a character in Hello
l is a character in Hello
o is a character in Hello
looks like o is not a character in Hello
```

Remember in for/else and while/else:
else is a form of “do this when the loop ends”

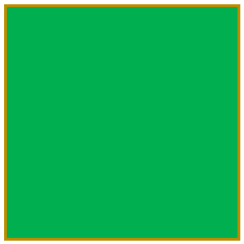
for loops



Let's rewrite our for loop a little

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

for loops

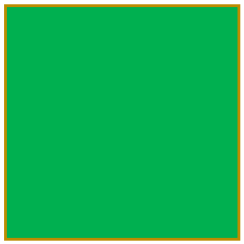


Let's rewrite our for loop a little

```
val = "e"
item_to_check = "Hello"
for val in item_to_check:
    print(val, "is a character in Hello")
else:
    print("looks like", val, "is not a character in Hello")
```

```
item_to_check = "hello"
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

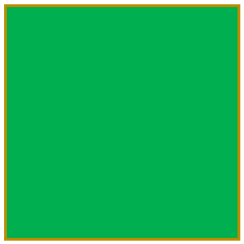
for loops



Let's rewrite our for loop a little

```
item_to_check = "hello"
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

for loops



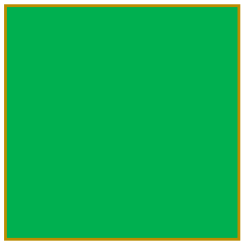
Let's rewrite our for loop a little

```
item_to_check = "hello"
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

Output:

```
h is a value in hello
e is a value in hello
l is a value in hello
l is a value in hello
o is a value in hello
looks like o is the last value in hello
```

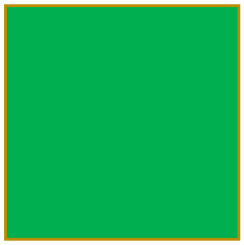
for loops



Will this work for other inputs?

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

for loops



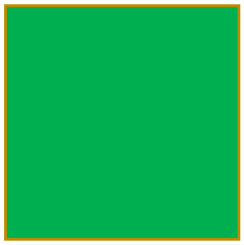
Will this work for other inputs?

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

Output:

```
1 is a value in [1, 2, 3, 4, 5]
2 is a value in [1, 2, 3, 4, 5]
3 is a value in [1, 2, 3, 4, 5]
4 is a value in [1, 2, 3, 4, 5]
5 is a value in [1, 2, 3, 4, 5]
looks like 5 is the last value in [1, 2, 3, 4, 5]
```

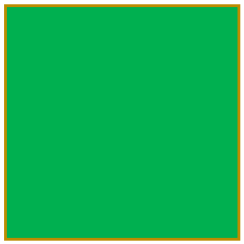
for loops



And what about reverse?

```
reverse = []
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check)
    reverse = [val] + reverse
else:
    print("looks like", val, "is not the last value in", reverse)
```

for loops



And what about reverse?

```
reverse = []
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check)
    reverse = [val] + reverse
else:
    print("looks like", val, "is not the last value in", reverse)
```

Output:

```
1 is a value in [1, 2, 3, 4, 5]
2 is a value in [1, 2, 3, 4, 5]
3 is a value in [1, 2, 3, 4, 5]
4 is a value in [1, 2, 3, 4, 5]
5 is a value in [1, 2, 3, 4, 5]
looks like 5 is not the last value in [5, 4, 3, 2, 1]
```

for loops

As our if and while are both Booleans and we could replace if with while in some cases

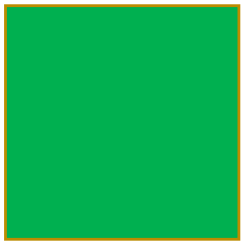
Can we replace if and while with for?

Let's see:

Okay, we can replace if with for

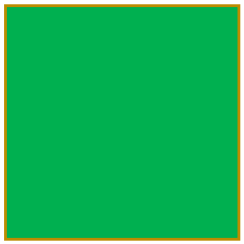
But what about while…

while from week 5



```
# WHILE EXAMPLES FROM WEEK 5
i = 0
while i < 10:
    print(i)
    i += 1
```

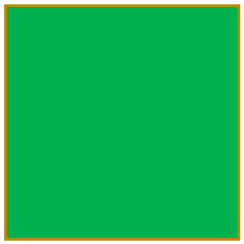
while from week 5



```
# WHILE EXAMPLES FROM WEEK 5
i = 0
while i < 10:
    print(i)
    i += 1
```

```
i = 0
for i < 10:
    print(i)
    i += 1
```

while from week 5



```
# WHILE EXAMPLES FROM WEEK 5
i = 0
while i < 10:
    print(i)
    i += 1
```

```
i = 0
for i < 10:
    print(i)
    i += 1
```

No

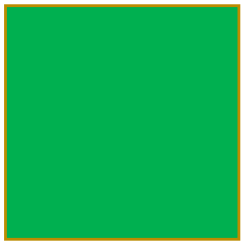
Semester 1 revision

Week 6

Lecture 17

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

for range



Similar to slicing, range consists of:

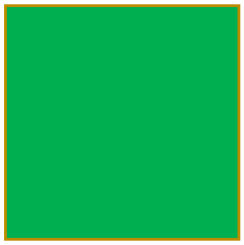
`range(start , end , step)`

`start` is zero by default

Returned values do NOT include the `end` value!!!!

`step` is the values to *step* over the list

for range



range(end)

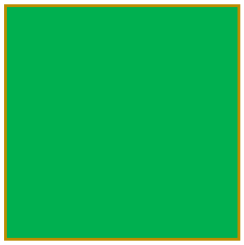
for val in range(5):

print(val)

Output:

What is the ouput?

for range



range(end)

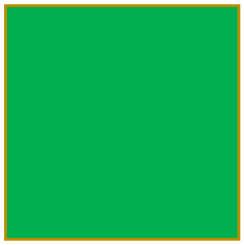
for val in range(5):

print(val)

Output:

0 1 2 3 4

for range



range(start , end)

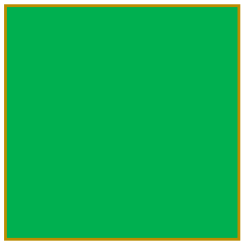
for val in range(1 , 5):

print(val)

Output:

What is the ouput?

for range



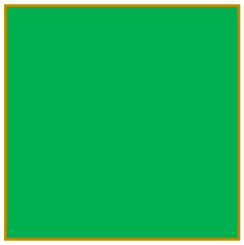
```
range(start , end)
```

```
for val in range(1 , 5):
    print(val)
```

Output:

1 2 3 4

for range



range(start , end , step)

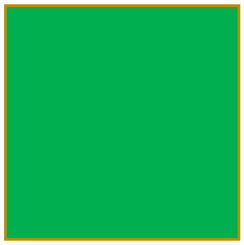
for val in range(1 , 5 , 2):

print(val)

Output:

What is the ouput?

for range



range(start , end , step)

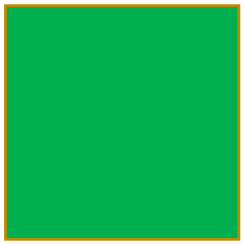
for val in range(1 , 5 , 2):

print(val)

Output:

1 3

for range



range(start , end , step)

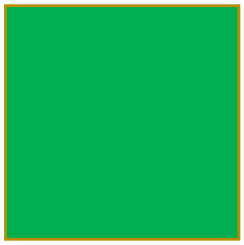
Negative steps

for val in range(-1 , -5 , -2):
print(val)

Output:

What is the ouput?

for range



range(start , end , step)

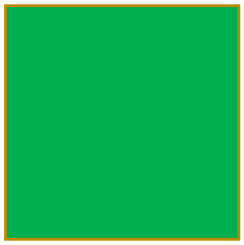
Negative steps

for val in range(-1 , -5 , -2):
print(val)

Output:

-1 -3

for range



range(start , end , step)

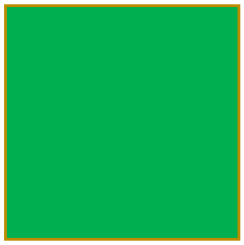
Negative steps

for val in range(10 , 4 , -2):
print(val)

Output:

What is the ouput?

for range



```
range(start , end , step)
```

Negative steps

```
for val in range(10 , 4 , -2):
    print(val)
```

Output:

10 8 6

Semester 1 revision

Week 6

Lecture 18

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Recap

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

Indexing example 1

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

Output:

Indexing example 1

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check)
else:
    print("looks like", val, "is the last value in", item_to_check)
```

Output:

```
1 is a value in [1, 2, 3, 4, 5]
2 is a value in [1, 2, 3, 4, 5]
3 is a value in [1, 2, 3, 4, 5]
4 is a value in [1, 2, 3, 4, 5]
5 is a value in [1, 2, 3, 4, 5]
looks like 5 is the last value in [1, 2, 3, 4, 5]
```

Indexing example 1 (rewrite)

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check,
          "at index", item_to_check.index(val))
else:
    print("looks like", val, "is the last value in",
          item_to_check, "at index", item_to_check.index(val))
```

Output:

Indexing example 1 (rewrite)

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check,
          "at index", item_to_check.index(val))
else:
    print("looks like", val, "is the last value in",
          item_to_check, "at index", item_to_check.index(val))
```

Output:

Why should we not use index() for this??

Indexing example 1 (rewrite)

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check,
          "at index", item_to_check.index(val))
else:
    print("looks like", val, "is the last value in",
          item_to_check, "at index", item_to_check.index(val))
```

Output:

<list_name>.index(<value>) returns
the index number <value> first occurs at, in the List

Indexing example 1 (rewrite)

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for val in item_to_check:
    print(val, "is a value in", item_to_check,
          "at index", item_to_check.index(val))
else:
    print("looks like", val, "is the last value in",
          item_to_check, "at index", item_to_check.index(val))
```

Output:

```
1 is a value in [1, 2, 3, 4, 5] at index 0
2 is a value in [1, 2, 3, 4, 5] at index 1
3 is a value in [1, 2, 3, 4, 5] at index 2
4 is a value in [1, 2, 3, 4, 5] at index 3
5 is a value in [1, 2, 3, 4, 5] at index 4
looks like 5 is the last value in [1, 2, 3, 4, 5] at index 4
```

Indexing example 2

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for index in range(len(item_to_check)):
    print(item_to_check[index], "is a value in", item_to_check)
else:
    print("looks like", item_to_check[index],
          "is the last value in", item_to_check)
```

Output:

Indexing example 2

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for index in range(len(item_to_check)):
    print(item_to_check[index], "is a value in", item_to_check)
else:
    print("looks like", item_to_check[index],
          "is the last value in", item_to_check)
```

Output:

```
1 is a value in [1, 2, 3, 4, 5]
2 is a value in [1, 2, 3, 4, 5]
3 is a value in [1, 2, 3, 4, 5]
4 is a value in [1, 2, 3, 4, 5]
5 is a value in [1, 2, 3, 4, 5]
looks like 5 is the last value in [1, 2, 3, 4, 5]
```

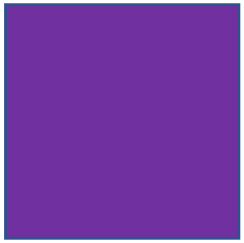
Indexing example 2 (rewrite)

We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for index in range(len(item_to_check)):
    print(item_to_check[index], "is a value in",
          item_to_check, "at index", index)
else:
    print("looks like", item_to_check[index],
          "is the last value in", item_to_check, "at index", index)
```

Output:

Indexing example 2 (rewrite)



We now have 2 ways of getting information from an indexable structure, e.g., list, string, tuple,…

```
item_to_check = [1, 2, 3, 4, 5]
for index in range(len(item_to_check)):
    print(item_to_check[index], "is a value in",
          item_to_check, "at index", index)
else:
    print("looks like", item_to_check[index],
          "is the last value in", item_to_check, "at index", index)
```

Output:

```
1 is a value in [1, 2, 3, 4, 5] at index 0
2 is a value in [1, 2, 3, 4, 5] at index 1
3 is a value in [1, 2, 3, 4, 5] at index 2
4 is a value in [1, 2, 3, 4, 5] at index 3
5 is a value in [1, 2, 3, 4, 5] at index 4
looks like 5 is the last value in [1, 2, 3, 4, 5] at index 4
```

Recap

Let’s learn a third way:

```
item_to_check = [1, 2, 3, 4, 5]
for index, val in enumerate(item_to_check):
    print(val, "is a value in", item_to_check, "at index", index)
else:
    print("looks like", val,
          "is the last value in", item_to_check, "at index", index)
```

enumerate – returns the index and value at each value from an indexable structure

Reduces the need to use <input>.index() or <input>[index] to get the index/value from <input>

Recap

Let's learn a third way:

```
item_to_check = [1, 2, 3, 4, 5]
for index, val in enumerate(item_to_check):
    print(val, "is a value in", item_to_check, "at index", index)
else:
    print("looks like", val,
          "is the last value in", item_to_check, "at index", index)
```

Output:

```
1 is a value in [1, 2, 3, 4, 5] at index 0
2 is a value in [1, 2, 3, 4, 5] at index 1
3 is a value in [1, 2, 3, 4, 5] at index 2
4 is a value in [1, 2, 3, 4, 5] at index 3
5 is a value in [1, 2, 3, 4, 5] at index 4
looks like 5 is the last value in [1, 2, 3, 4, 5] at index 4
```

while from week 5

```
def reverse(a_list):

    reversed_list = []
    i = 0
    while i < len(a_list):
        j = 0
        while j < len(a_list[i]):
            reversed_list = [a_list[i][j]] + reversed_list
            j += 1
        i += 1

    return reversed_list


my_list = [[1, 2, 3, 4], [5, 6, 7, 8, 9]]
print(reverse(my_list))
```

We have seen nested while loops

Summer 2019 exam question

```
def Func(s1, s2):
    # summer 2018 - Q2.a - 10% - 22.5 marks
    # write this using while loops
    r = []
    for e1 in s1:
        for e2 in s2:
            if e1 == e2:
                r += [e1]
                break
    return r
```

To explain nested for loops
I'm going to use an exam question

Summer 2019 exam question

```
def Func(s1, s2):
    # summer 2018 - Q2.a - 10% - 22.5 marks
    # write this using while loops
    r = []
    for e1 in s1:
        for e2 in s2:
            if e1 == e2:
                r += [e1]
                break
    return r
```

```
print(Func([2, 8, 7, 2, 9], [2, 2, 9, 0, 9]))
```

Summer 2019 exam question

```
def Func(s1, s2):
    # summer 2018 - Q2.a - 10% - 22.5 marks
    # write this using while loops
    r = []
    for e1 in s1:
        for e2 in s2:
            if e1 == e2:
                r += [e1]
                break
    return r
```

```
print(Func([2, 8, 7, 2, 9], [2, 2, 9, 0, 9]))
```

```
# [2, 2, 9]
```

Summer 2019 exam question

```python
def Func(s1, s2):
    # summer 2018 - Q2.a - 10% - 22.5 marks
    # write this using while loops
    r = []
    for e1 in s1:
        for e2 in s2:
            if e1 == e2:
                r += [e1]
                break
    return r
```

Summer 2019 exam question

```python
def Func(s1, s2):
    # summer 2018 - Q2.a - 10% - 22.5 marks
    # write this using while loops
    r = []
    for e1 in s1:
        for e2 in s2:
            if e1 == e2:
                r += [e1]
                break
    return r
```

```python
def Func_while(s1, s2):
    # summer 2018 - Q2.a - 10% - 22.5 marks
    # write this using while loops
    r = []
    i = 0
    while i < len(s1):
        j = 0
        while j < len(s2):
            if s1[i] == s2[j]:
                r += [s1[i]]
                break
            j += 1
        i += 1
    return r
```

Semester 1 revision

Week 7

No Lectures

Semester 1 revision

Week 8

No Lectures

Semester 1 revision

Week 9

Lecture 25

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Lab Recap

Let us look at some of the issues that keep popping up in the lab submissions:

Let’s start with Boolean Functions:

Lab Recap

This is a Boolean functions from Lab 7, called is_wet()

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

The instruction in the Lab was that the function is called is_wet() defined without a parameter and returns True or False

So the Code is good ☺

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

No, 'a' can only be one of two choices.

So we only need to check for one of them

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

No, 'a' can only be one of two choices.

So we only need to check for one of them

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    else:
        return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

We can actually remove the else.

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

We can actually remove the else.

If a is 'yes', then True is returned

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

This works for return, not for print(), why??

We can actually remove the else.

If a is 'yes', then True is returned

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

We can actually remove the else.

If a is 'no', then False is returned

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True

    return False
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

Actually, we can remove the 'if', if we set our choice to True or False

```
def is_wet():
    a = random.choice([True, False])
    return a
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

```
def is_wet():
    a = random.choice(['True', 'False'])
    return a
```

Note: this is not the same, these are strings with values 'True' and 'False'

```
def is_wet():
    a = random.choice([True, False])
    return a
```

Lab Recap

But the code can be refined, so let's start with the returned choice

```
import random


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False
```

Do we need to recheck if 'a' is no ?

Actually, we can remove the 'a' variable, if we just return the random choice

```
def is_wet():
    return random.choice([True, False])
```

Lab Recap

Before I move away from this function, lets look at functions with the same name

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Lab Recap

Before I move away from this function, lets look at functions with the same name

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Note in the second definition, I reversed the returned values

Lab Recap

Before I move away from this function, lets look at functions with the same name

```
def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Note in the second definition, I reversed the returned values

Python will only remember the last time a function, or a variable, was defined, so here 'yes' will always return False

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

```
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

Remember variables with the same text but with different upper and lower are different

```
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

Remember variables with the same text but with different upper and lower are different

```
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

So are functions with slightly different names, they are also different

Lab Recap

Also, if the function is to be called ‘is_wet()’, then that is its name, not any kind of variation.

Remember variables with the same text but with different upper and lower are different

```
def is_wet1():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return True
    elif a == 'no':
        return False


def is_wet2():
    a = random.choice(['yes', 'no'])
    if a == 'yes':
        return False
    elif a == 'no':
        return True
```

So are functions with slightly different names, they are also different

In Android phones, the onCreate() function is called to launch the app (ish), so if you wrote onCreate1() it will not work

Lab Recap

Now, let's return to our revised is_wet() Boolean Function

```
def is_wet():
    return random.choice([True, False])
```

This will return True or False

So how do we check for this?

Using if and while?

Let's see….

Lab Recap

Using if and while?

```
def is_wet():
    return random.choice([True, False])
```

```
if is_wet() == True:
    print("OMG - mom is not going to be happy....")

while is_wet() == True:
    print("OMG - mom is really not going to be happy....")
```

Lab Recap

Using if and while?

```
def is_wet():
    return random.choice([True, False])
```

```
if is_wet() == True:
    print("OMG - mom is not going to be happy....")

while is_wet() == True:
    print("OMG - mom is really not going to be happy....")
```

```
# output
# OMG - mom is not going to be happy....
# OMG - mom is really not going to be happy....
# OMG - mom is really not going to be happy....
# OMG - mom is really not going to be happy....
# OMG - mom is really not going to be happy....
```

Lab Recap

Let's look at the if condition statement

```
if is_wet() == True:
    print("OMG - mom is not going to be happy....")

while is_wet() == True:
    print("OMG - mom is really not going to be happy....")
```

We are checking if the returned value from is_wet() is True

```
if is_wet() == True:
```

I see where you are coming from,

but let's look at this a little bit more

Lab Recap

First off, see the yellow underline…

Python is not happy with this code and issues a warning:

```
if is_wet() == True:
```

E712 comparison to True should be

'if cond is True:' or 'if cond:'

Lab Recap

First off, see the yellow underline…

Python is not happy with this code and issues a warning:

```
if is_wet() == True:
```

E712 comparison to True should be

'if cond is True:' or 'if cond:'

It is the same in PyCharm – white underline:

```
if is_wet()==True:
```

Expression can be simplified

Lab Recap

So, let's run the code and see what happens…

```
if is_wet() == True:
```

Let's say is_wet() returns True:

```
if is_wet() == True:
```

Lab Recap

So, let's run the code and see what happens…

```
if is_wet() == True:
```

Let's say is_wet() returns True:

```
if is_wet() == True:
```

True

Lab Recap

So, let's run the code and see what happens…

```
if is_wet() == True:
```

Let's say is_wet() returns True:

```
if is_wet() == True:
```

True

Now we are checking if True == True

Lab Recap

So, let's run the code and see what happens…

```
if is_wet() == True:
```

Let's say is_wet() returns True:

```
if is_wet() == True:
```
True

Now we are checking if True == True

```
if is_wet() == True:
```
True

So all good, right?? Well…

Lab Recap

So, let's run the code and see what happens…

```
if is_wet() == True:
```

Let's say is_wet() returns True:

```
if is_wet() == True:
```
True

Now we are checking if True == True

```
if is_wet() == True:
```
True

So all good, right?? Well…

Do you see that the value of is_wet() is actually the value we use?

Lab Recap

Let’s see what happens when is_wet() is False:

```
if is_wet() == True:
```

is_wet() returns False:

```
if is_wet() == True:
```

False

Lab Recap

Let’s see what happens when is_wet() is False:

```
if is_wet() == True:
```

is_wet() returns False:

```
if False == True:
```

Now we are checking if False == True

```
if False:
```

Lab Recap

Let's see what happens when is_wet() is False:

```
if is_wet() == True:
```

is_wet() returns False:

```
if is_wet() == True:
```
False

Now we are checking if False == True

```
if is_wet() == True:
```
False

Again, do you see that the value of is_wet() is actually the value we use?

Lab Recap

So, when we have a conditional statement, using a Boolean function, we just need to call the function:

```
if is_wet() == True:
```

Becomes:

```
if is_wet():
```

or

```
while is_wet():
```

Lab Recap

One last item on is_wet()

I am also seeing this kind of code:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Here we have two calls to is_wet() and if we assume each returns a different value, we could get:

Lab Recap

One last item on is_wet()

I am also seeing this kind of code:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

False

True

Here we have two calls to is_wet() and if we assume each returns a different value, we could get:

Lab Recap

One last item on is_wet()

I am also seeing this kind of code:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

False

True

Here we have two calls to is_wet() and if we assume each returns a different value, we could get:

Nothing being printed…

Lab Recap

Take a moment and think how this should be rewritten:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Lab Recap

Take a moment and think how this should be rewritten:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

If we take is_wet() and save in a variable,

at least we only call is_wet() once

```
is_gizmo_wet = is_wet()
if is_gizmo_wet == True:
    print("Gizmo is a triplet")
elif is_gizmo_wet == False:
    print("Gizmo is fine")
```

Lab Recap

Take a moment and think how this should be rewritten:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

Actually once we define the variable, it is a Boolean

So we can use the variable as our conditional statement

```
is_gizmo_wet = is_wet()
if is_gizmo_wet:
    print("Gizmo is a triplet")
else:
    print("Gizmo is fine")
```

Lab Recap

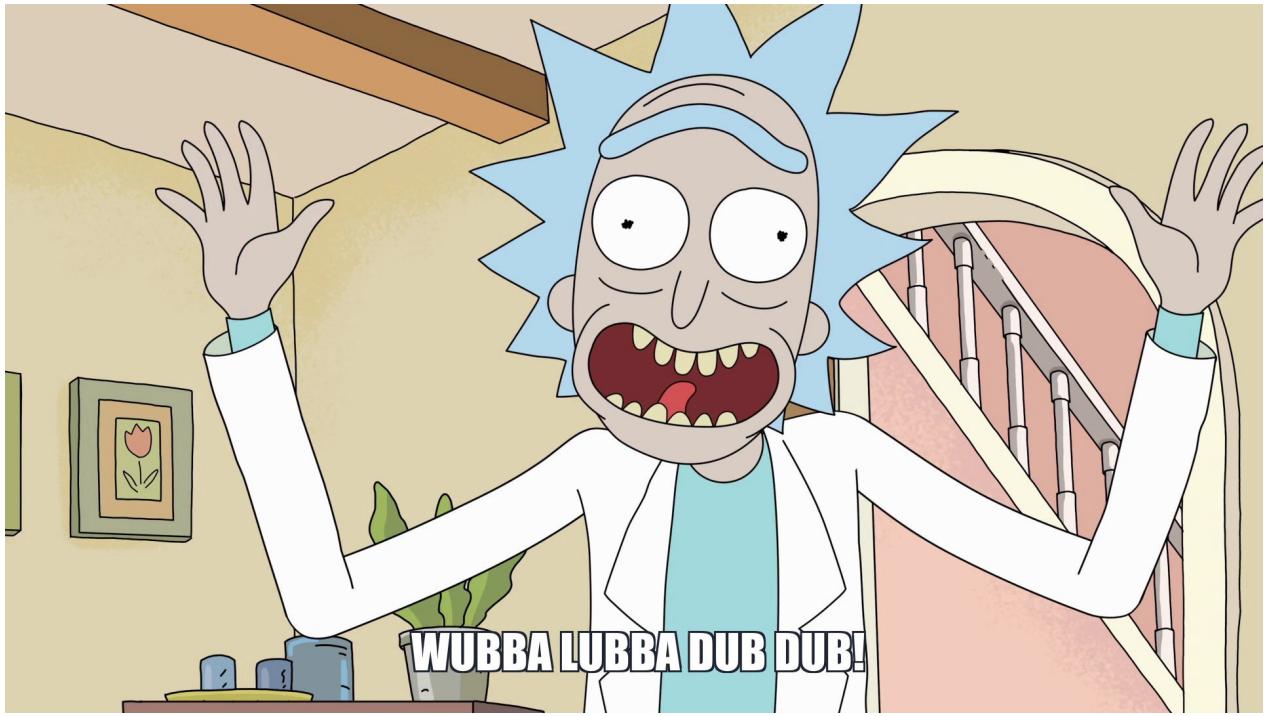
Take a moment and think how this should be rewritten:

```
if is_wet() == True:
    print("Gizmo is a triplet")
elif is_wet() == False:
    print("Gizmo is fine")
```

But the best thing to do, is just use is_wet()

```
if is_wet():
    print("Gizmo is a triplet")
else:
    print("Gizmo is fine")
```

One call, no additional variables, and easy to follow



UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh