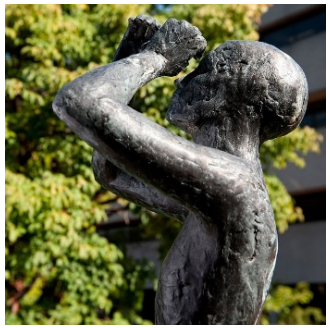


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

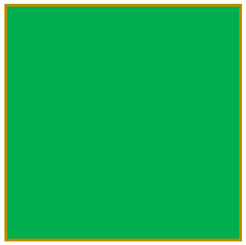
A TRADITION OF INDEPENDENT THINKING

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

☺

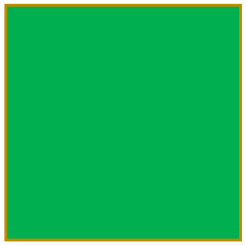
Semester 1 revision

Week 9

Lecture 26

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Dictionaries



Dictionary defined using curly brackets:

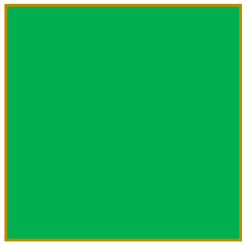
```
my_dictionary = {} # empty dictionary
```

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

key is a unique item

value can be a recurring item

Dictionaries



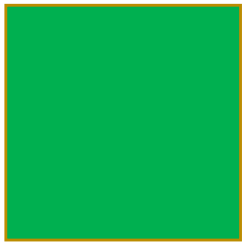
Dictionary defined using curly brackets:

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'])
```

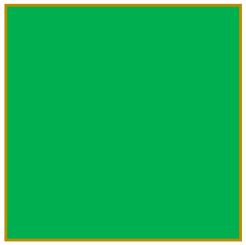
And get values, using my_dictionary.values()

```
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:

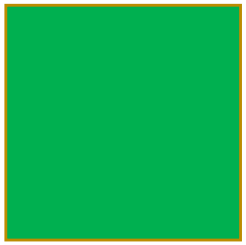
```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

```
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:

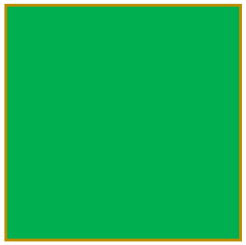
```
my_dictionary = {"February": "Spring", "March": "Spring", "April": "Spring"}
```

```
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

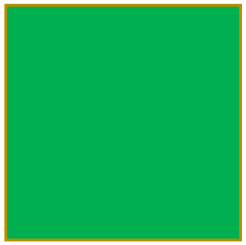
```
my_dictionary["February"] = "Swinter"
print(my_dictionary["February"])
# Swinter
```

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>)

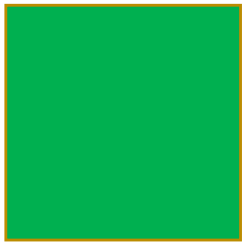
We can add to a List – mutable – <list>[4] = value

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

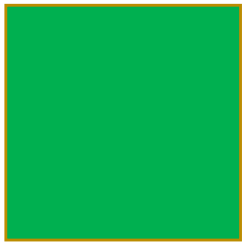
We can not use count as the keys are unique

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???

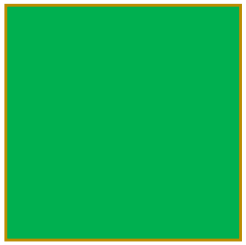
Semester 1 revision

Week 9

Lecture 27

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

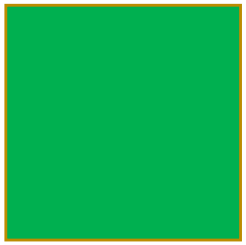
Dictionary Example



Some more Dictionary examples:

```
# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}

#  month of birth
mob = {'January': ['aishling', 'tim'], 'February': ['bob']}
```

Dictionary Example

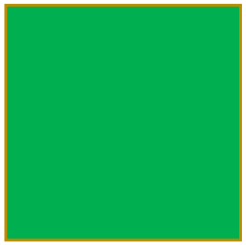


Some more Dictionary examples:

```
# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}

#  month of birth
mob = {'January': ['aishling', 'tim'], 'February': ['bob']}
```

```
# print month:
for m in mob:
    print(m)
#  prints
# January
# February

for m in mob.keys():
    print(m)
#  prints
# January
# February
```

Dictionary Example



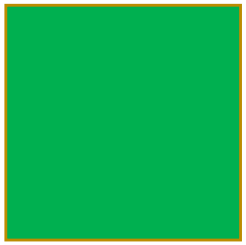
Some more Dictionary examples:

```
# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}

#  month of birth
mob = {'January': ['aishling', 'tim'], 'February': ['bob']}
```

```
# print month:
for m in mob:
    print(m)
#  prints
# January
# February

for m in mob.keys():
    print(m)
#  prints
# January
# February
```

```
# print people in month:
for m in mob:
    print(mob[m])
# prints
# ['aishling', 'tim']
# ['bob']

# print people in month:
for v in mob.values():
    print(v)
# prints
# ['aishling', 'tim']
# ['bob']
```

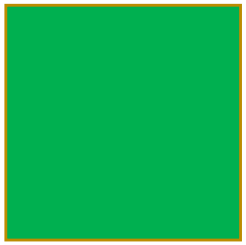
Dictionary Example



Some more Dictionary examples:

```
# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}

#  month of birth
mob = {'January': ['aishling', 'tim'], 'February': ['bob']}

months = {1: 'January', 2: 'February'}


def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months as keys and
    a list of names born in those months as values
    '''
    return
```

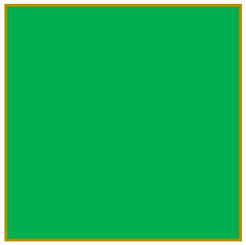
Dictionary Example



Some more Dictionary examples:

```
# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}

#  month of birth
# mob = {'January': ['aishling', 'tim'], 'February': ['bob']}


def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    months = {1: 'January', 2: 'February'}

    return


#  month of birth
mob = to_month(dob)
```

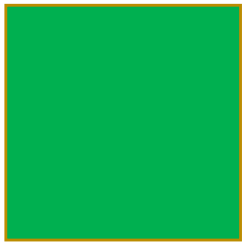
Dictionary Example



Some more Dictionary examples:

```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    months = {1: 'January', 2: 'February'}

    return



# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# None
```

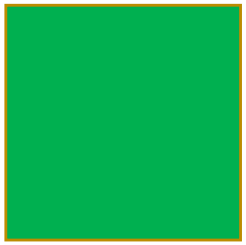
Dictionary Example



Some more Dictionary examples:

```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    months = {1: 'January', 2: 'February'}

    return dob


# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
```

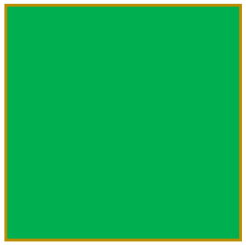
Dictionary Example



Some more Dictionary examples:

```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    months = {1: 'January', 2: 'February'}
    mob_dict = {}

    return mob_dict



# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# {}
```

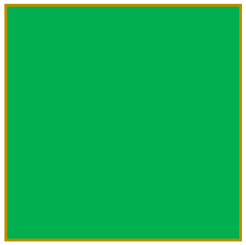
Dictionary Example



Some more Dictionary examples:

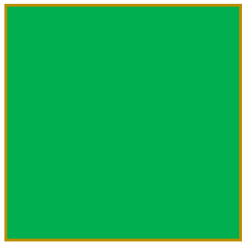
```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    months = {1: 'January', 2: 'February'}
    mob_dict = {}
    for key, val in dob.items():
        print(key, val)

    return mob_dict


# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# aishling 1/1/1995
# bob 2/2/1996
# tim 17/1/1995
# {}
```

Dictionary Example



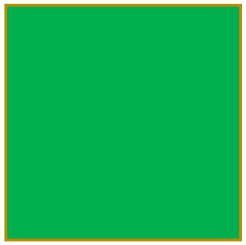
```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    mob_dict = {}
    for key, val in dob.items():
        month = get_month(val)
        print(month)

    return mob_dict


# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# January
# February
# January
# {}
```

Dictionary Example



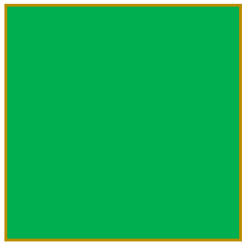
```
def get_month(val):
    '''
    take a dob in european format - '1/1/1995'
    return integer for month - 1 -> 'January', etc.
    '''
    months = {1: 'January', 2: 'February'}

    slash_first_index = val.index("/")  # 1
    val = val[slash_first_index+1:]  # '1/1995'

    slash_second_index = val.index("/")  # 1
    val = val[:slash_second_index]  # '1'

    val = int(val)  # 1

    return months[val]  # 'January'
```

Dictionary Example



```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    mob_dict = {}
    for key, val in dob.items():
        month = get_month(val)
        mob_dict[month] = [key]

    return mob_dict



# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# {'January': ['tim'], 'February': ['bob']}
```

Dictionary Example



```
def to_month(dob):
    '''
    dob - take in a dictionary of peoples date of birth
    return - a dictionary of months keys and
    a list of names born in those months as values
    '''
    mob_dict = {}
    for key, val in dob.items():
        month = get_month(val)
        if month in mob_dict:
            mob_dict[month].append(key)
        else:
            mob_dict[month] = [key]

    return mob_dict


# date of birth
dob = {'aishling': '1/1/1995', 'bob': '2/2/1996', 'tim': '17/1/1995'}
#  month of birth
mob = to_month(dob)
print(mob)
# {'January': ['aishling', 'tim'], 'February': ['bob']}
```

Lab Recap

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

Let's start with break and return:

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

The instruction was to create a function called index which takes 2 parameters (list, value) and return the first index of value or -1 if index does not exist in the list

so this code is fine ☺

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

But, I've seen people write list.index(value) in their code ☹

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

Let’s look at the return -1

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

Let's look at the return -1

This will never be returned – why??

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

Let's look at the return -1

This will never be returned – why??

1. If value in list – returns True – then return -1 is never called

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

Let's look at the return -1

This will never be returned – why??

2. The else: is indented too far – it should line up to if

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

Let's look at the return -1

This will never be returned – why??

3. Not an issue here, but a call to break will not call else after while…

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1
        else:
            return -1
```

So let's move the return -1

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1

    return -1
```

So let's move the return -1

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1

    return -1
```

So let's move the return -1

Remember if value in list – return -1 never called

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    if value in list:
        i = 0
        while i < len(list):
            if list[i] == value:
                return i
                break
            i += 1

    return -1
```

So let's move the return -1

Remember if value in list – return -1 never called

Lab Recap

This is the index function from Lab 6

So let's move the return -1

Remember if value not in list – return -1 is called

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    # if value in list:
    i = 0
    while i < len(list):
        if list[i] == value:
            return i
            break
        i += 1

    return -1
```

Actually we don't need the if value in list (but it does speed up our code…)

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    # if value in list:
    i = 0
    while i < len(list):
        if list[i] == value:
            return i
            break
        i += 1

    return -1
```

Finally, let's look at the return and break

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    # if value in list:
    i = 0
    while i < len(list):
        if list[i] == value:
            return i
            break
        i += 1

    return -1
```

return ends the function (may return a value to the calling code)

Break stops and exits the loop (ignoring else)

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    # if value in list:
    i = 0
    while i < len(list):
        if list[i] == value:
            return i
            break
        i += 1

    return -1
```

return ends the function (may return a value to the calling code)

Break stops and exits the loop (ignoring else)

So what's the problem here?

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    # if value in list:
    i = 0
    while i < len(list):
        if list[i] == value:
            return i
            break
        i += 1

    return -1
```

return ends the function (may return a value to the calling code)

Break stops and exits the loop (ignoring else)

So what's the problem here? – break is never called

Lab Recap

This is the index function from Lab 6

```
def index(list, value):
    # if value in list:
    i = 0
    while i < len(list):
        if list[i] == value:
            return i
            # break
        i += 1

    return -1
```

return ends the function (may return a value to the calling code)

Break stops and exits the loop (ignoring else)

So what's the problem here? – so remove break

Lab Recap

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

Let's now look at shadowing:

Shadowing is redefining a variable/function name which masks the original use of the name

Lab Recap

Index() from Lab 6

```
def index(list, value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(list):
        # if <value> and <list> index i are the same
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the counter
        i += 1
    return index(value)
```

Lab Recap

```
def index(list, value):
    # set cou
    i = 0
    # loop ov
    while i <
        # if
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the counter
        i += 1
    return index(value)
```

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.

```
def index(list, value):
```

Lab Recap

```
def index(list, value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(list):
        # if <value> and <list> index i are the same
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the counter
        i += 1
    return index(value)
```

Lab Recap

```
def index(list, value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(list):
        # if <value> and <list> index i are the same
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the cou
        i += 1
    return index(value)
```

Return first index of value.

Raises ValueError if the value is not present.

Lab Recap

```
def index(list, value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(list):
        # if <value> and <list> index i are the same
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the counter
        i += 1
    return index(value)
```

index(list, value)

```
def index(list, value):
```

```
list = [1, 2, 12]
```

Lab Recap

If I change the list parameter to my_list

```
def index(my_list, value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(list):
        # if <value> and <list> index i are the same
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the counter
        i += 1
    return index(value)
```

No errors!!!

Lab Recap

If I change the value parameter to my_value

```
def index(my_list, my_value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(list):
        # if <value> and <list> index i are the same
        if list[i] == value:
            return list.index(value)
        else:
            print('-1')
        # increment the counter
        i += 1
    return index(value)
```

Errors!!!

Lab Recap

Rewrite the function

```
def index(my_list, value):
    # set counter
    i = 0
    # loop over the length of the <list>
    while i < len(my_list):
        # if <value> and <list> index i are the same
        if my_list[i] == value:
            return i
        # increment the counter
        i += 1
    # value not in list, so return -1
    return -1
```

Return i, not list.index(value)

Move and return -1, not “-1”

Lab Recap

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

A few other things to keep in mind:

Lab Recap

Print is not return

Print is only used so we can see what a variable value is

Or what a value returned from a functions is

Lab Recap

```
def what_is_returned():
    var = 1


print(what_is_returned())
```

Lab Recap

```
def what_is_returned():
    var = 1


print(what_is_returned())
# prints 'None'
```

The Python None keyword is used to define a null value, or no value at all

None is not the same as 0, False, or an empty string

None is a datatype of its own (NoneType) and only None can be None

Lab Recap

```python
def what_is_returned():
    var = 1


print(what_is_returned())
# prints 'None'
```

```python
def what_is_returned():
    var = 1

    return


print(what_is_returned())
```

Lab Recap

```python
def what_is_returned():
    var = 1


print(what_is_returned())
# prints 'None'
```

```python
def what_is_returned():
    var = 1

    return


print(what_is_returned())
# prints 'None'
```

Lab Recap

```
def what_is_returned():
    var = 1

print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return

print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return var


print(what_is_returned())
```

Lab Recap

```
def what_is_returned():
    var = 1


print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return


print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return var


print(what_is_returned())
# prints 1
```

Lab Recap

```
def what_is_returned():
    var = 1


print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return


print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return var


print(what_is_returned())
# prints 1
```

```
def what_is_returned():
    var = 1

    return var


what_is_returned()
```

Lab Recap

```
def what_is_returned():
    var = 1


print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return


print(what_is_returned())
# prints 'None'
```

```
def what_is_returned():
    var = 1

    return var


print(what_is_returned())
# prints 1
```

```
def what_is_returned():
    var = 1

    return var


what_is_returned()
# nothing is presented on terminal
# or in the run screen
```

Lab Recap

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

Let's now look at lists and slicing:

Lab Recap

start = [:index]

is not

start = list[:index]

Also, remember

list[:index] stops at the value of index-1

Lab Recap

```
my_list = [4, 5, 6, 7]
max_index = 3
sub_list = [:max_index]
sub_list = my_list[:max_index]
```

Lab Recap

```
my_list = [4, 5, 6, 7]
max_index = 3
sub_list = [:max_index]
sub_list = my_list[:max_index]
```

```
my_string = "hello"
max_index = 3
sub_string = [:max_index]
sub_string = my_string[:max_index]
```

Lab Recap

```
my_list = [4, 5, 6, 7]
max_index = 3
sub_list = [:max_index]
sub_list = my_list[:max_index]
```

```
my_string = "hello"
max_index = 3
sub_string = [:max_index]
sub_string = my_string[:max_index]
```

```
my_string = "hello"
max_index = 3
# sub_string = [:max_index]
sub_string = my_string[:max_index]
print(sub_string)
# hel
```

Lab Recap

list1.append(list2)

Will add list2 as a single element to list1:

[1, 2, 3].append([2, 3, 4])

[1, 2, 3, [2, 3, 4]]

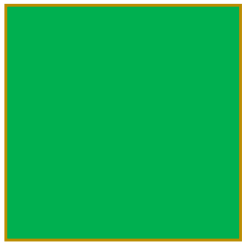
Semester 1 revision

Week 11

Lecture 31

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

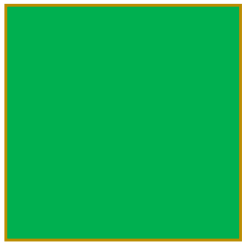
Scope



for loop producing a list from 0 to 9

```
# a list from 0 to 9
x = []
for i in range(10):
    x.append(i)
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

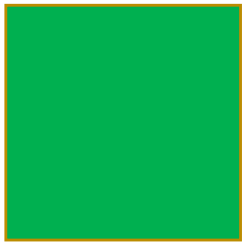
Scope



for loop producing a list from 0 to 9

```
# a list from 0 to 9
x = []
for i in range(10):
    x.append(i)
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
```

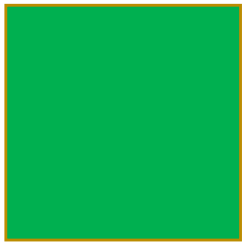
Scope



for loop producing a list from 0 to 9

```
# a list from 0 to 9
x = []
for i in range(10):
    x.append(i)
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
# the next value of i is 9
```

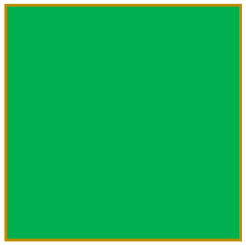
Scope



while loop producing a list from 0 to 9

```
# a list from 0 to 9
x = []
i = 0
while i < 10:
    x.append(i)
    i += 1
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

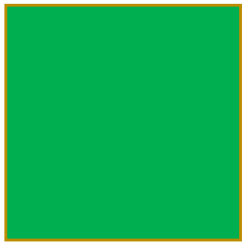
Scope



while loop producing a list from 0 to 9

```
# a list from 0 to 9
x = []
i = 0
while i < 10:
    x.append(i)
    i += 1
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
```

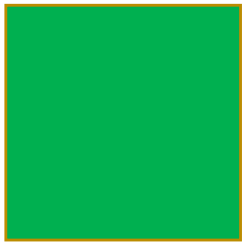
Scope



while loop producing a list from 0 to 9

```
# a list from 0 to 9
x = []
i = 0
while i < 10:
    x.append(i)
    i += 1
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
# the next value of i is 10
```

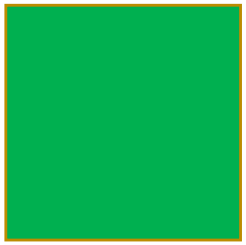
Scope



Function producing a list from 0 to 9

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

Scope

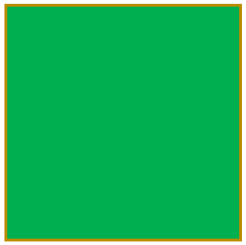


Function producing a list from 0 to 9

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

```
print(return_list(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Scope

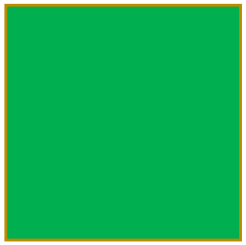


Function producing a list from 0 to 9

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

```
print(return_list(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is",
```

Scope

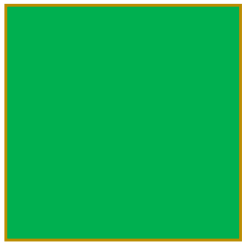


Function producing a list from 0 to 9

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

```
print(return_list(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
```

Scope



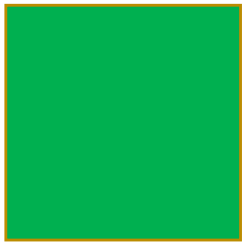
Function producing a list from 0 to 9

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

```
print(return_list(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
```

```
NameError: name 'i' is not defined
```

Scope

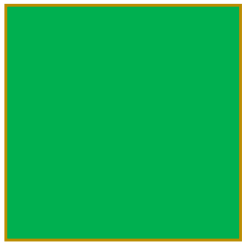


Function producing a list from 0 to 9

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

```
i = 0
print(return_list(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
```

Scope

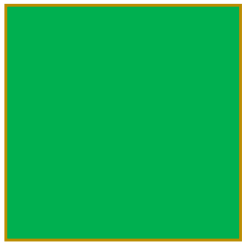


Function producing a list from 0 to 9

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

```
i = 0
print(return_list(10))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
# the next value of i is 0
```

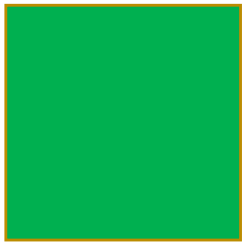
Scope



Function producing a list from 0 to 9

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

Scope

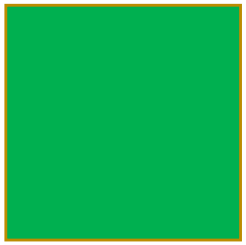


Function producing a list from 0 to 9

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

```
i = 0
x, i = return_list(10)
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
```

Scope

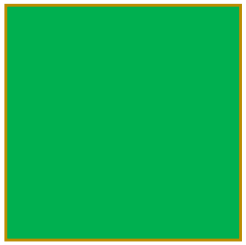


Function producing a list from 0 to 9

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

```
i = 0
x, i = return_list(10)
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
# the next value of i is 9
```

Scope

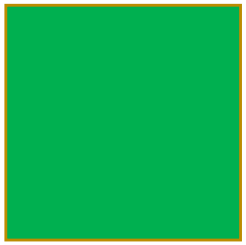


Function producing a list from 0 to 9

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

```
# i = 0
x, i = return_list(10)
print(x)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i)
# the next value of i is 9
```

Scope

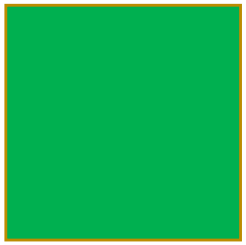


Function producing a list from 0 to 9

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

```
# i = 0
my_list, counter = return_list(10)
print(x)
#
print("the next value of i is", i)
#
```

Scope

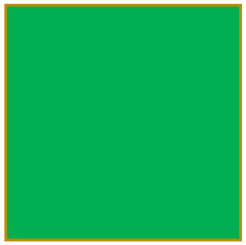


Function producing a list from 0 to 9

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

```
x_2, i_2 = return_list(10)
print(x_2)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i_2)
# the next value of i is 9
```

Scope

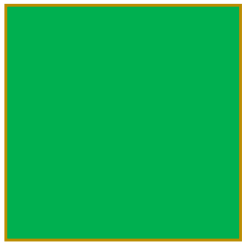


Function producing a list from 0 to 9

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

```
x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
# ???
print("the next value of i is", i_2)
# the next value of i is 9
```

Scope

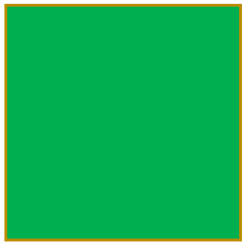


Function producing a list from 0 to 9

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

```
x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i_2)
# the next value of i is 9
```

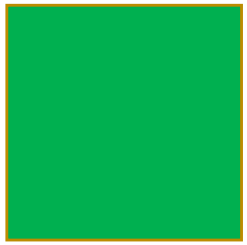
Scope



Function producing a list from 0 to 9

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

Scope



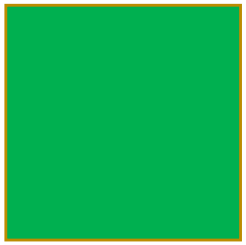
Function producing a list from 0 to 9

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

No errors for ‘x’

```
x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
# ???
print("the next value of i is", i_2)
# the next value of i is 9
```

Scope

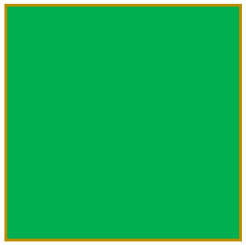


Function producing a list from 0 to 9

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

```
x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
# [1, 2, 3, 4, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i_2)
# the next value of i is 9
```

Scope

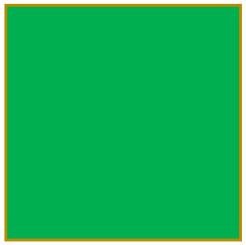


Function producing a list from 0 to 9

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

```
# x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
#
print("the next value of i is", i_2)
#
```

Scope



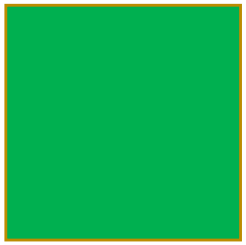
Function producing a list from 0 to 9

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

Errors for 'x'

```
# x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
#
print("the next value of i is", i_2)
#
```

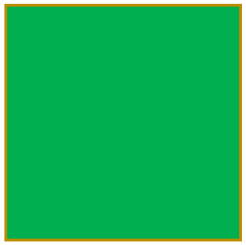
Scope



Function producing a list from 0 to 9

```
x = [1, 2, 3, 4]


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


# x = [1, 2, 3, 4]
x_2, i_2 = return_list(10)
print(x_2)
# [1, 2, 3, 4, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("the next value of i is", i_2)
# the next value of i is 9
```

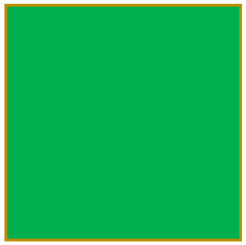
Scope



Function producing a list from 0 to 9

```
x = [1, 2, 3, 4]
number = 5


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


# x = [1, 2, 3, 4]
x_2, i_2 = return_list()
print(x_2)
# ????
print("the next value of i is", i_2)
# ????
```

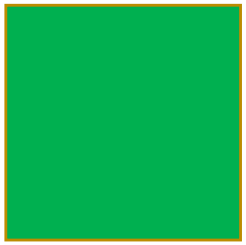
Scope



Function producing a list from 0 to 9

```
x = [1, 2, 3, 4]
number = 5


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


# x = [1, 2, 3, 4]
x_2, i_2 = return_list()
print(x_2)
# [1, 2, 3, 4, 0, 1, 2, 3, 4]
print("the next value of i is", i_2)
# the next value of i is 4
```

Scope



So, variable values can be seen in a function when not defined in a function

But not normally when defined in a function when called outside of a function

Scope is a good concept to help understand how variables can impact on your code

Always make sure you pass the correct values as parameters, and return variable as needed…

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh