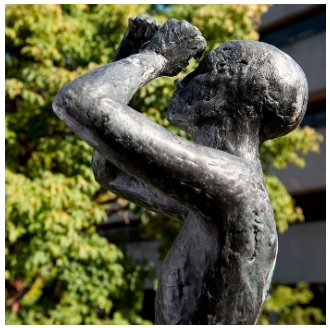


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

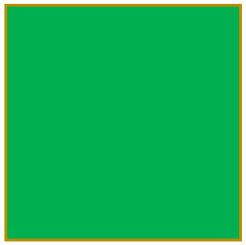
A TRADITION OF INDEPENDENT THINKING

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

☺

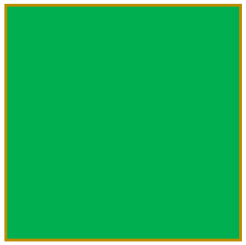
Semester 1 revision

Week 4

Lecture 10

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Tuple recap



We can now create tuples – 2 different ways

We can get a value based on index number – tuple[index]

We can get the length of a tuple – len(tuple)

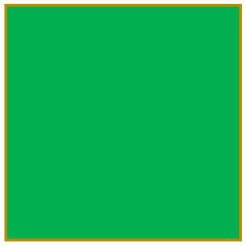
We can't add to a tuple – immutable – tuple[4] = value

We can count how many times a value appears in a tuple – tuple.count(value)

And we can get the index based on value (throws exception) if the value does not exist in the tuple – tuple.index(value)

Oh and we can use - if value in tuple:

Tuple recap



But…

Tuples are immutable

Once we create them we can't change them

Which of little benefit to us when we want a dynamic system

So we need a structure like Tuples, but which can change

So let's look at our list of Data types

Tuple / List

- If we look at Tuple and List in our Data Types table

Type	Example
Tuple	x = ("Ed", "Edd", "Eddy", 2009)
List	x = ["Ed", "Edd", "Eddy", 2009]

- We can see that the content of both types is identical
- The only physical difference being the ( ) of Tuple
- And [ ] of List
- But List is mutable, it's content can be changed…

List

- In List we can use all the functions we have seen 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>:

Let's look at some examples:

Lists – examples...

Create a List using list()

Create a List from a Tuple using list()

```
great_show = list(["Ed", "Edd", "Eddy", 2009])
print(great_show)
great_show = list(("Ed", "Edd", "Eddy", 2009))
print(great_show)

# output
# ['Ed', 'Edd', 'Eddy', 2009]
# ['Ed', 'Edd', 'Eddy', 2009]
```

Round Brackets

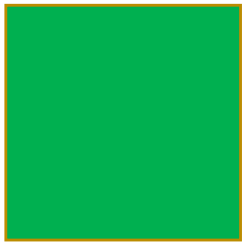
Lists – examples...

Get the index value from a List using <list_name>[<index>]

```
great_show = ["Ed", "Edd", "Eddy", 2009]
print(great_show[1])

# output
# Edd
```

Square Brackets

Lists – examples...



Add an element to the List

If the index does not exist, we get an error

```
great_show = ["Ed", "Edd", "Eddy", 2009]
great_show[4] = "plank"

# output
# IndexError: list assignment index out of range
```

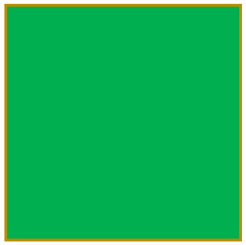
Index 4 does not exist

Lists – examples...

Here we append() to the end of the List

```
great_show = ["Ed", "Edd", "Eddy", 2009]
great_show.append("plank")
print(great_show)
great_show[4] = "Jonny"
print(great_show[4])
print(great_show)

# output
# ['Ed', 'Edd', 'Eddy', 2009, 'plank']
# Jonny
# ['Ed', 'Edd', 'Eddy', 2009, 'Jonny']
```

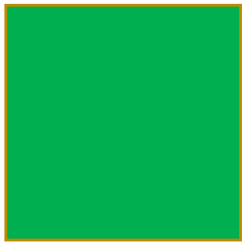
Lists – examples...



Here I use append() or index allocation depending on List length

```
great_show = ["Ed", "Edd", "Eddy", 2009]
index_to_add = 4
if len(great_show) <= index_to_add:
    great_show.append("Jonny")
    print("I use append")
else:
    great_show[index_to_add] = "Jonny"
    print("I use index allocation")

print(great_show)

# output
# I use append
# ['Ed', 'Edd', 'Eddy', 2009, 'Jonny']
```

Lists – examples...



As CS indexing starts at zero

len() will always return one more than the highest index number

```
great_show = ["Ed", "Edd", "Eddy", 2009]
              0      1       2       3
len(great_show) -> 4
```

Index number of value 2009 is 3

So max index is one less than len()

```
len(great_show )-1
```

Remember this…

Lists – examples...

Here I use append or index allocation depending on List length

Change index to add to 2

```
great_show = ["Ed", "Edd", "Eddy", 2009]
index_to_add = 2
if len(great_show) <= index_to_add:
    great_show.append("Jonny")
    print("I use append")
else:
    great_show[index_to_add] = "Jonny"
    print("I use index allocation")

print(great_show)

# output
# I use index allocation
# ['Ed', 'Edd', 'Jonny', 2009]
```

Lists – examples...

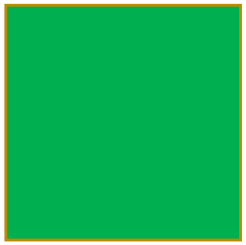
Here I use append or index allocation depending on List length

Change index to add to 2

```
great_show = ["Ed", "Edd", "Eddy", 2009]
index_to_add = 2
if len(great_show) <= index_to_add:
    great_show.append("Jonny")
    print("I use append")
else:
    great_show.insert(index_to_add, "Jonny")
    print("I use index allocation")

print(great_show)

# output
# I use index allocation
# ['Ed', 'Edd', 'Jonny', 2009]
```

I can also use great_show.insert(index, value)

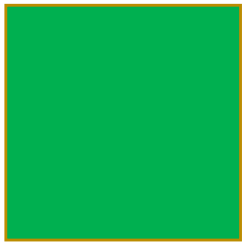
Lists – examples...



<list_name>.count(<value>) returns
how many times <value> occurs in the List

```
great_show = ["Ed", "Ed", "Ed", 2009]
print(str(great_show[0]) + " occurs "
      + str(great_show.count(great_show[0]))
      + " time(s) in this list")

# output
# Ed occurs 3 time(s) in this list
```

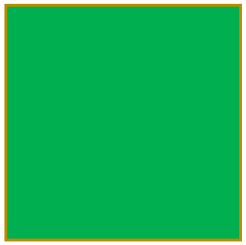
Lists – examples...



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

```
great_show = ["Ed", "Ed", "Ed", 2009]
print("Ed occurs at index "
      + str(great_show.index("Ed"))
      + " in this list")

# output
# Ed occurs at index 0 in this list
```

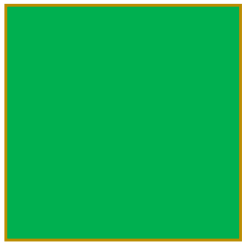
Lists – examples...



Looking for a <value> using <list_name>.index(<value>) that does not occurs in the List will cause an error

```
great_show = ["Ed", "Ed", "Ed", 2009]
# print("ed occurs at index " + str(great_show.index("ed")) + " in this list")

# output
# ValueError: 'ed' is not in list
```

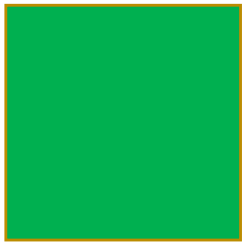
Lists – examples...



We can use try/except to catch these errors

```
try:
    great_show = ["Ed", "Ed", "Ed", 2009]
    print("ed occurs at index "
          + str(great_show.index("ed"))
          + " in this list")
except Exception as e:
    print("ed does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ['Ed', 'Ed', 'Ed', 2009]
```

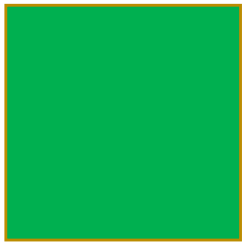
Lists – examples...



Finally, we can use in and not in

```
great_show = ["Ed", "Ed", "Ed", 2009]
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs in "
          + str(great_show))
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

if value_to_find not in great_show:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ['Ed', 'Ed', 'Ed', 2009]
# ed does not occurs in ['Ed', 'Ed', 'Ed', 2009]
```

Lists – examples...

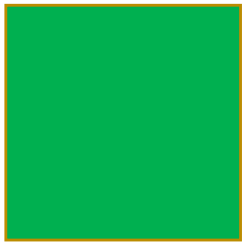


So, we've seen all the functionality common between tuple and list

Let's look at new functionality for list

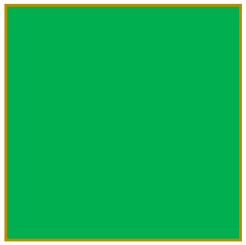
[1,2,3] * 3 -> [1,2,3,1,2,3,1,2,3]

Lists – examples...



We can use negative indexes to get values from the end of a list

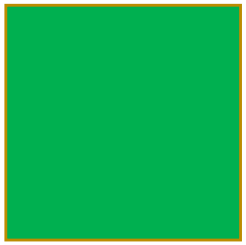
Lists – examples...



We can use negative indexes to get values from the end of a list

my_list = [“a”, “b”, “c”]

Lists – examples...

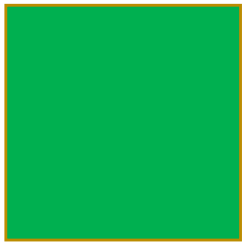


We can use negative indexes to get values from the end of a list

my_list = [“a”, “b”, “c”]

my_list[2] -> “c”

Lists – examples...



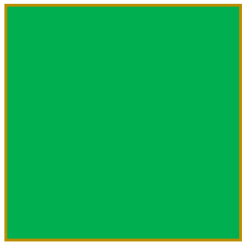
We can use negative indexes to get values from the end of a list

my_list = [“a”, “b”, “c”]

0 1 2

my_list[2] -> “c”

Lists – examples...



We can use negative indexes to get values from the end of a list

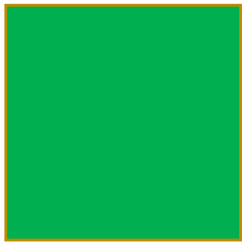
my_list = [“a”, “b”, “c”]

0 1 2

my_list[2] -> “c”

my_list[-1] -> “c”

Lists – examples...



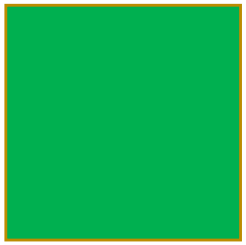
We can use negative indexes to get values from the end of a list

-3	-2	-1
my_list = [“a”, “b”, “c”]		
0	1	2

my_list[2] -> “c”

my_list[-1] -> “c”

Lists – examples...



We can use + operator to concatenate lists together

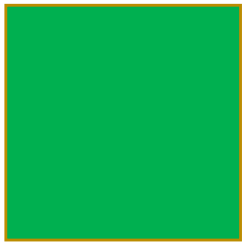
my_list_1 = [“a”, “b”, “c”]

my_list_2 = [“d”, “e”, “f”]

my_list_1 = my_list_1 + my_list_2

my_list_1 -> [“a”, “b”, “c”, “d”, “e”, “f”]

Lists – examples...



We can use += operator to concatenate lists together

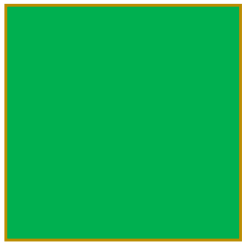
my_list_1 = [“a”, “b”, “c”]

my_list_2 = [“d”, “e”, “f”]

my_list_1 += my_list_2

my_list_1 -> [“a”, “b”, “c”, “d”, “e”, “f”]

Lists – examples...



We can use sort() to sort the contents of a list from lowest to highest value

my_list_1 = [“a”, “B”, “7”]

my_list_1.sort()

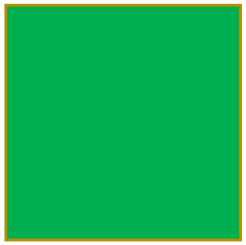
my_list_1 -> [“7”, “B”, “a”]

my_list_2 = [“9”, “i”, “g”]

my_list_2.sort()

my_list_1 -> [“9”, “g”, “i”]

Lists – examples...



We can use remove(value) to remove the first occurrence of a value from a list

my_list_1 = [“a”, “B”, “7”]

my_list_1.remove(“7”)

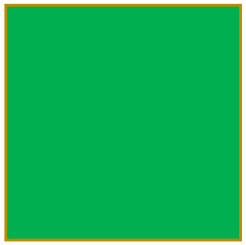
my_list_1 -> [“a”, “B”]

my_list_2 = [“9”, “i”, “g”]

my_list_2. remove (“7)

Error - ValueError: list.remove(x): x not in list

Lists – examples...



We can use del list [index] to remove the index from a list

my_list_1 = [“a”, “B”, “7”]

del my_list_1[1]

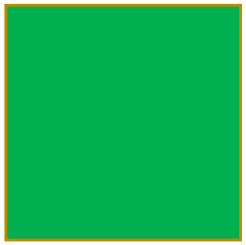
my_list_1 -> [“a”, “7”]

my_list_2 = [“9”, “i”, “g”]

del my_list_2[0]

my_list_2 -> [“i”, “g”]

Lists – examples...



We can use reverse() to reverse the list

my_list_1 = [“a”, “B”, “7”]

my_list_1.reverse()

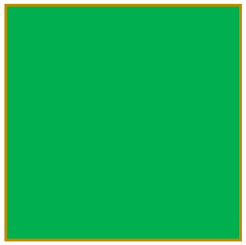
my_list_1 -> [“7”, “B”, “a”]

my_list_2 = [“9”, “i”, “g”]

my_list_2. reverse()

my_list_1 -> [“g”, “i”, “9”]

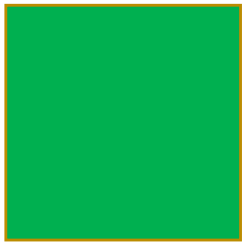
Lists – examples...



We can use min()/max() operator to get min and max values in a list

```
my_list_1 = [“a”, “B”, “7”]
min_val = min(my_list_1)
min_val -> “7”

max_val = max(my_list_1)
max_val -> “a”
```

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

0 1 2 3 4

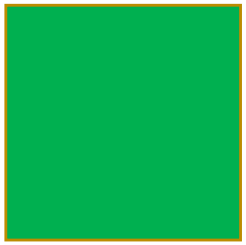
new_list = my_list_1[1 : 4]

new_list -> [“B”, “7”, "d"]

list[*start* : *end*]

Returned list does NOT include the end index!!!!

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

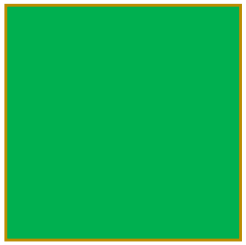
0 1 2 3 4

new_list = my_list_1[: 4]

new_list -> [“a”, “B”, “7”, "d"]

If we don’t specify the start index
0 is used by default

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, “d”, “4"]

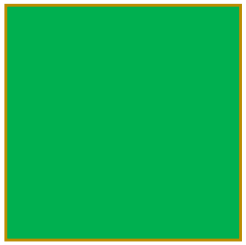
0 1 2 3 4

new_list = my_list_1[1:]

new_list -> [“B”, “7”, "d", "4"]

If we don’t specify the end index
len(list) is used by default

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, “d”, “4"]

0 1 2 3 4

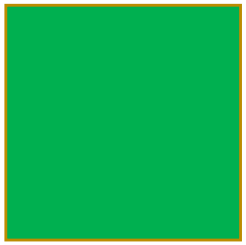
new_list = my_list_1[1:]

new_list -> [“B”, “7”, "d", "4"]

If we don’t specify the end index
len(list) is used by default

Why don’t we use len(list)-1??

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

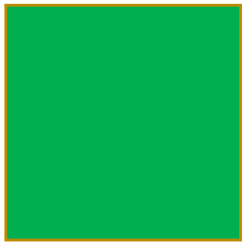
0 1 2 3 4

new_list = my_list_1[:]

new_list -> [“a”, “B”, “7”, "d", "4"]

If we don’t specify the start or end index
The entire list is returned by default

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

-5 -4 -3 -2 -1

new_list = my_list_1[: -1]

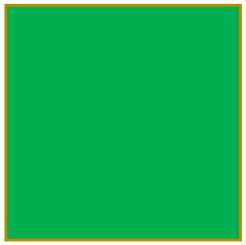
new_list -> [“a”, “B”, “7”, "d"]

We can also use the negative indexes

new_list = my_list_1[: -4]

new_list -> [“a”]

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

-5 -4 -3 -2 -1

new_list = my_list_1[0 : -1]

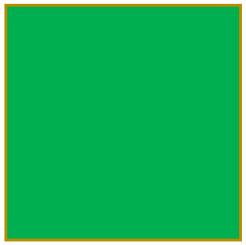
new_list -> [“a”, “B”, “7”, "d"]

We can actually use any combinations of positive and negative values

new_list = my_list_1[2: -2]

new_list -> [“7”]

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

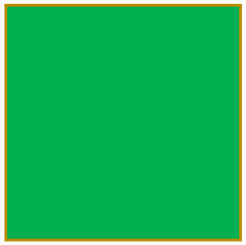
-5 -4 -3 -2 -1

new_list = my_list_1[3 : 1]

new_list -> []

If start is larger than end
You get an empty list

Lists – examples...



Slicing – extracting a subset of the list values

my_list_1 = [“a”, “B”, “7”, "d”, "4"]

-5 -4 -3 -2 -1

new_list = my_list_1[3 : 1]

new_list -> []

But we can add a third step value to step over the list

new_list = my_list_1[3 : 1 : -1]

new_list -> [“d”, ”7”]

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append()

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort()

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value)

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index]

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse()

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list)

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

Operators - *

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

Operators - *, +

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

Operators - *, +, +=

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

Operators - *, +, +=

Slicing – list[start:end]

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

Operators - *, +, +=

Slicing – list[start:end], list[start:end:step]

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

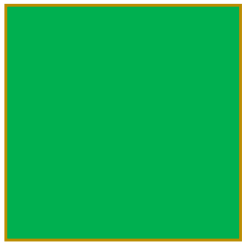
Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

Operators - *, +, +=

Slicing – list[start:end], list[start:end:step]

Negative indexing – list[-2]

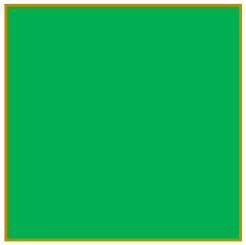
Slicing



Does Slicing only work for Lists?

```
my_list_1 = ["a", "B", "7", "d", "4"]
new_list = my_list_1[2:]
print(new_list)

# output
# ['7', 'd', '4']
```

Slicing

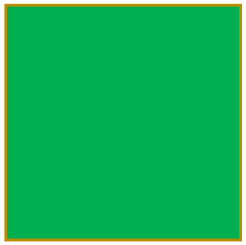


Does Slicing only work for Lists?

```
my_tuple_1 = ("a", "B", "7", "d", "4")
new_tuple = my_tuple_1[2:]
print(new_tuple)

# output
# ('7', 'd', '4')
```

Works for Tuples…

Slicing

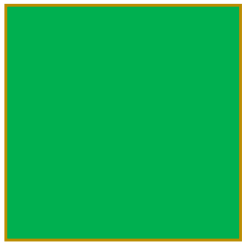


Does Slicing only work for Lists?

```
string_1 = "hello world"
new_string = string_1[2:]
print(new_string)

# output
# llo world
```

Works for Strings…

Slicing



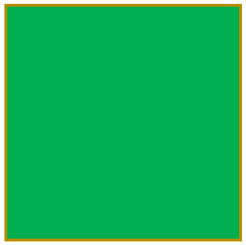
Does Slicing only work for Lists?

```
int_1 = 123456
new_int = int_1[2:]
print(new_int)

# output
# TypeError: 'int' object is not subscriptable
```

Does not work for Integers….

What about Floats?

Slicing



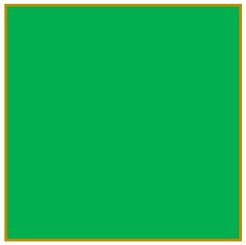
Does Slicing only work for Lists?

```
float_1 = 123.456
new_float = float_1[2:]
print(new_float)

# output
# TypeError: 'float' object is not subscriptable
```

Does not work for Integers or Floats…

But….

Slicing

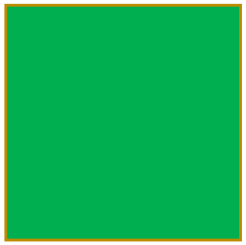


Does Slicing only work for Lists?

```
int_1 = str(123456)
new_int = int_1[2:]
print(new_int)

# output
# 3456
```

Cast int to String, and then slicing works ☺

Slicing



Does Slicing only work for Lists?

```
float_1 = str(123.456)
new_float = float_1[2:]
print(new_float)

# output
# 3.456
```

Cast float to String, and then slicing works ☺

List Recap

In List we can use all the functions we have seen for Tuple, plus we can use:

Functions - list.append(), list.sort(), list.remove(value), del list[index], list.reverse(), min/max(list), insert(index,value)

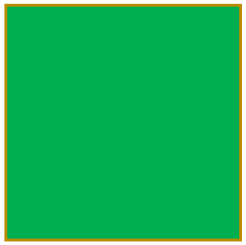
Operators - *, +, +=

Negative indexing – list[-2]

Slicing – list[start:end], list[start:end:step]

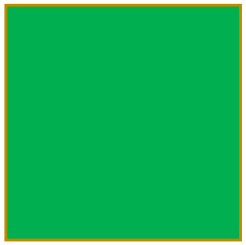
Slicing works for string, list, tuple but not float or int (cast…)

Slicing



So Slicing work for a host of different Data Types?

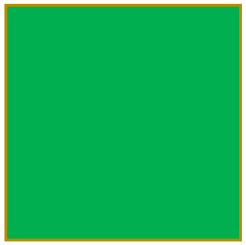
Slicing



So Slicing work for a host of different Data Types?

When the Data Type does not support indexing

Slicing

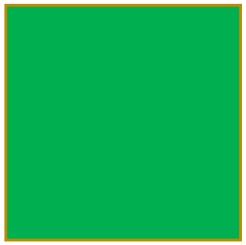


So Slicing work for a host of different Data Types?

When the Data Type does not support indexing

We can cast to a Data Type that does support indexing

Slicing



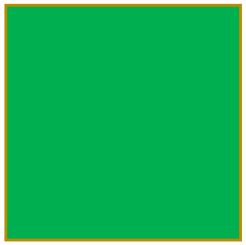
So Slicing work for a host of different Data Types?

When the Data Type does not support indexing

We can cast to a Data Type that does support indexing

Do our Slicing

Slicing



So Slicing work for a host of different Data Types?

When the Data Type does not support indexing

We can cast to a Data Type that does support indexing

Do our Slicing

And then cast back to the original Data Type

☺

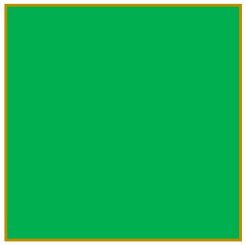
Semester 1 revision

Week 5

Lecture 13

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

While

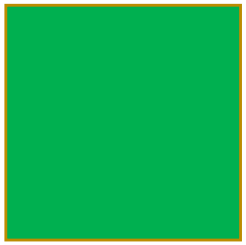


While loop – sample output

```
# empty Harry Potter list
HP_list = []

# ask a question
HP_item = input(
    "Tell me something you liked from the Harry Potter movies: (press enter to stop): ")

while HP_item != "":
    # append to list
    HP_list.append(HP_item)
    # ask the question again
    HP_item = input(
        "Tell me something you liked from the Harry Potter movies: (press enter to stop): ")

print(HP_list)
```

```
Tell me something you liked from the Harry Potter movies: (press enter to stop): harry
Tell me something you liked from the Harry Potter movies: (press enter to stop): hermonie
Tell me something you liked from the Harry Potter movies: (press enter to stop): ron
Tell me something you liked from the Harry Potter movies: (press enter to stop):
['harry', 'hermonie', 'ron']
```

While



We now 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

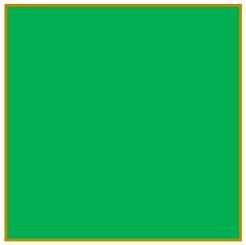
While the condition remains True,

execute the statement block

Remember we need some way to make the condition False

Otherwise it becomes an infinite loop…

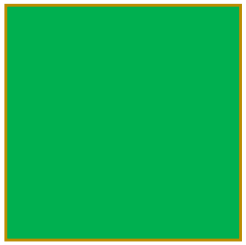
While



Let’s look at a very common example - output

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

print("Phew. The While has stopped")

# Output
# 0
# 1
# 2
# 3
# 4
# 5
# 6
# 7
# 8
# 9
# Phew. The While has stopped
```

While



If we change our code ever so slightly

```
i = 0
while i < 10:
    print(i, end=" ")
    i += 1

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

We can print the output on one line

```
0 1 2 3 4 5 6 7 8 9
Phew. The While has stopped
```

While

Let’s look at a very common mistake

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

If we forget to increment our counte

```
0
0
0
0
0
0
0
0
0
0
0
0
0
```

While

If we change our code ever so slightly

```
i = 0
while i < 10:
    print(i, end=" ")
    i += 2

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

# Output
# 0 2 4 6 8
# Phew. The While has stopped
```

If we change what we add to our counter,
we can print only even numbers

While

If we change our code ever so slightly

```
i = 1
while i < 10:
    print(i, end=" ")
    i += 2

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

# Output
# 1 3 5 7 9
# Phew. The While has stopped
```

And if we change the initial value of our counter we can print only odd numbers

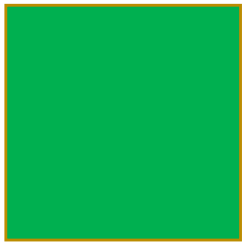
Semester 1 revision

Week 5

Lecture 14

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

While



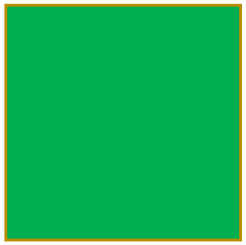
Let’s see if we can write some code to convert a word/phrase into snake case…

“hello world” => “h_e_l_l_o_w_o_r_l_d”

Let’s ask the user for a phase

And use a while loop for the conversion

While



Snake case …

```
word = input("Please input a word/phrase >>> ")
word_size = len(word)
i = 0
while i < word_size:
    print(word[i], end="_")
    i += 1
print()

# Output
# Please input a word/phrase >>> hello world
# h_e_l_l_o_ _w_o_r_l_d_
```

While

Snake case … let's try slightly different code

Let's remove the spacing

```
word = input("Please input a word/phrase >>> ")
word_size = len(word)
i = 0
while i < word_size:
    if word[i] != " ":
        print(word[i], end="_")
    i += 1
print()

# Output
# Please input a word/phrase >>> hello world
# h_e_l_l_o_w_o_r_l_d_
```

While

Snake case … let's try slightly different code

Let's remove the trailing underscore

```
word = input("Please input a word/phrase >>> ")
word_size = len(word)
i = 0
while i < word_size:

    if word[i] != " ":
        print(word[i], end="_")

    if i == word_size-1:
        print(word[i])

    i += 1

# Output
# Please input a word/phrase >>> hello world
# h_e_l_l_o_w_o_r_l_d_d
```

While

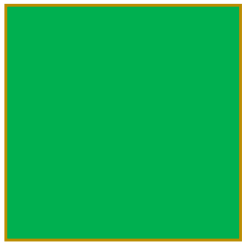
Snake case … let's try slightly different code

Let's remove the trailing underscore – let's use if/elif

Order matters!!!

```
word = input("Please input a word/phrase >>> ")
word_size = len(word)
i = 0
while i < word_size:

    if word[i] != " ":
        print(word[i], end="_")

    elif i == word_size-1:
        print(word[i])

    i += 1
print()

# Output
# Please input a word/phrase >>> hello world
# h_e_l_l_o_w_o_r_l_d_
```

While



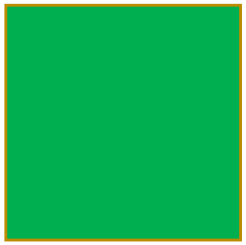
Snake case … let's try slightly different code

Let's remove the trailing underscore – let's use if/elif

Reverse the if and elif conditional statements

```
word = input("Please input a word/phrase >>> ")
word_size = len(word)
i = 0
while i < word_size:
    if i == word_size-1:
        print(word[i])

    elif word[i] != " ":
        print(word[i], end="_")

    i += 1

# Output
# Please input a word/phrase >>> hello world
# h_e_l_l_o_w_o_r_l_d
```

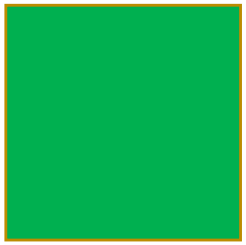
While



Few more examples:

```
# print values between 2 integers inclusive
def printValues(val1, val2):
    i = val1
    while i <= val2:
        print(i)
        i += 1


printValues(2, 5)

# Output
# 2
# 3
# 4
# 5
```

While

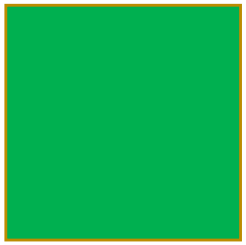


Few more examples:

```
# print values between 2 integers inclusive
# sum all the values together and print them out
def print_added_values(val1, val2):
    i = val1
    accum = 0
    while i <= val2:
        print(i)
        accum += i
        i += 1
    print("sum of all values:", accum)


print_added_values(2, 5)

# Output
# 2
# 3
# 4
# 5
# sum of all values: 14
```

An accumulator allows us to store a value over the duration of the while loop

While



Your Turn – Calculate the answer

```
x = 6
while x < 9:
    print(x)
    x = x + 2
print(x)
while x > 7:
    print(x)
    x = x - 1
```

While

Your Turn – Calculate the answer

```
x = 6
while x < 9:
    print(x)
    x = x + 2
print(x)
while x > 7:
    print(x)
    x = x - 1
```

```
6
8
10
10
9
8
```

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh