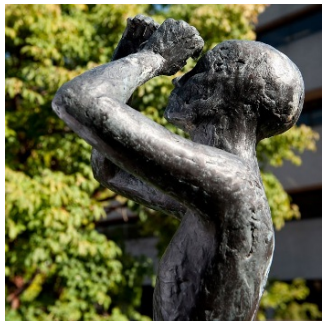


CS1117 – Introduction to Programming

Dr. Jason Quinlan,

School of Computer Science and Information Technology

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland

Coláiste na hOllscoile Corcaigh

Announcements

CS1117 DSA students

While your labs and lectures were cancelled yesterday

You still need to complete Lab 2

If I can not find an alternative lab this week, I will extend the submit by an extra week

Announcements

G20 Hard drive allocation

As we did yesterday in G20, please check your “quota”

Type quote into terminal

First number is what you have used

Second number is your total allocation

Difference is the remainder

if, if/else and elif Recap

- We introduced if conditional statements over ranges of values
  - num_demodog from 1 to 10
- We added checks to run code when a conditional statement is False

```
if (condition):
    run code if condition is True
else:
    run code if condition is False
```

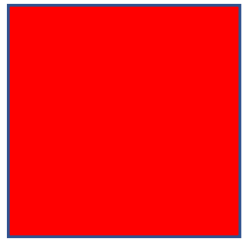
- And we added checks for multiple inputs using elif

```
if (condition1):
    run code if condition1 is True
elif (condition2):
    run code if condition2 is True
else:
    run code if both condition1 and condition2 are False
```

if, if/else and elif

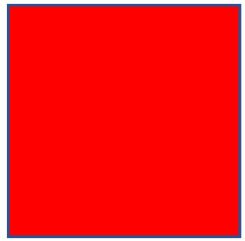
Live Coding Time…

Exception Handling



- We see lots and lots of errors when we code, e.g.,

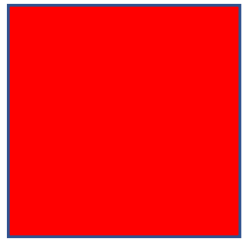
Exception Handling



- We see lots and lots of errors when we code, e.g.,

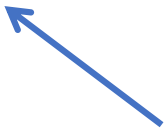
- TypeError: can only concatenate str (not "int") to str

Exception Handling



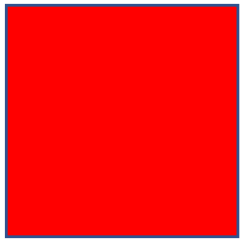
- We see lots and lots of errors when we code, e.g.,

- TypeError: can only concatenate str (not "int") to str



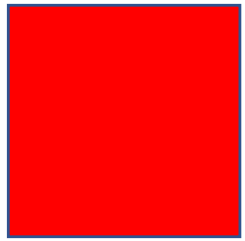
e.g., we have tried to print an int value that we have not cast to a string

Exception Handling

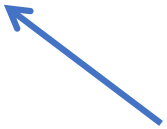


- We see lots and lots of errors when we code, e.g.,
- TypeError: can only concatenate str (not "ValueError") to str
- ValueError: invalid literal for int() with base 10: ‘g’

Exception Handling

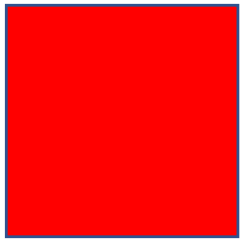


- We see lots and lots of errors when we code, e.g.,
- TypeError: can only concatenate str (not "ValueError") to str
- ValueError: invalid literal for int() with base 10: ‘g’



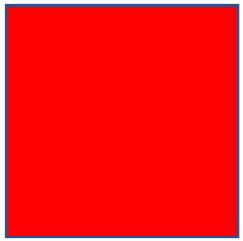
e.g., we have tried to cast an input string to an int, but it is not a int

Exception Handling

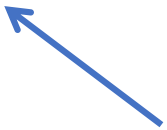


- We see lots and lots of errors when we code, e.g.,
- TypeError: can only concatenate str (not "ValueError") to str
- ValueError: invalid literal for int() with base 10: ‘g’
- NameError: name 'number' is not defined

Exception Handling

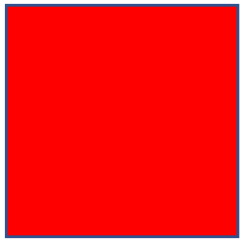


- We see lots and lots of errors when we code, e.g.,
- TypeError: can only concatenate str (not "ValueError") to str
- ValueError: invalid literal for int() with base 10: ‘g’
- NameError: name 'number' is not defined



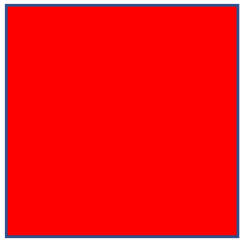
e.g., we have tried to call a variable but we have not yet assigned it a value

Exception Handling



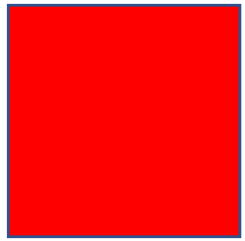
- We see lots and lots of errors when we code, e.g.,
- TypeError: can only concatenate str (not "ValueError") to str
- ValueError: invalid literal for int() with base 10: ‘g’
- NameError: name 'number' is not defined
- Each of these errors are bugs in our code, and while we will get lots of them as we code, we need to consider each and every one of them as we write

Exception Handling



- We see lots and lots of errors when we code, e.g.,
- TypeError: can only concatenate str (not "ValueError") to str
- ValueError: invalid literal for int() with base 10: ‘g’
- NameError: name 'number' is not defined
- Each of these errors are bugs in our code, and while we will get lots of them as we code, we need to consider each and every one of them as we write
- But….

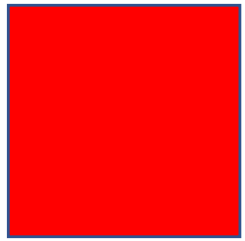
Exception Handling



- Python give us a very simple mechanism to catch these and all other errors:

- Exception Handling:

Exception Handling

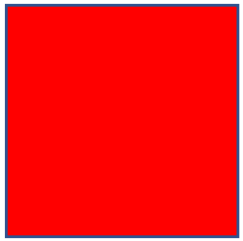


- Python give us a very simple mechanism to catch these and all other errors:

- Exception Handling:

- Instead of crashing, the exception handler prints a message indicating that there was a problem, or we can run some code.

Exception Handling



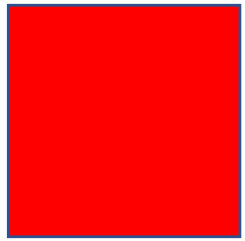
- Python give us a very simple mechanism to catch these and all other errors:

- Exception Handling:

- Instead of crashing, the exception handler prints a message indicating that there was a problem, or we can run some code.

- The `try…except` can be used to catch *any* kind of error and provide for a graceful exit.

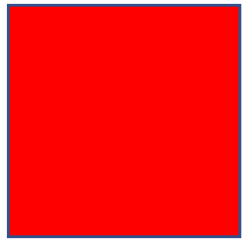
Exception Handling



The programmer can write code that catches and deals with errors that arise while the program is running, i.e., “Do these steps, and if any problem crops up, handle it this way.”

```
try:
      <body>
except:
      <handler>
```

Exception Handling



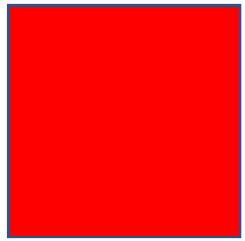
The programmer can write code that catches and deals with errors that arise while the program is running, i.e., “Do these steps, and if any problem crops up, handle it this way.”

```
try:
      <body>
except:
      <handler>
```

When Python encounters a try statement, it attempts to execute the statements inside the body.

If there is no error, control passes to the next statement after the try…except.

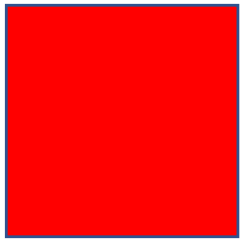
Exception Handling



So, let's look at an example

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Exception Handling

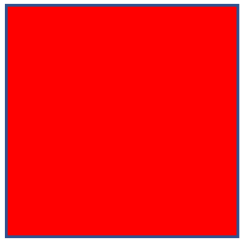


So, let's look at an example

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Here are our try/except

Exception Handling



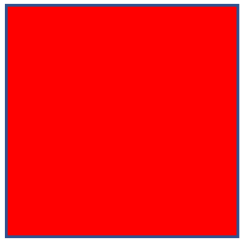
So, let's look at an example

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Here are our try/except

And our question is "enter a positive number"

Exception Handling

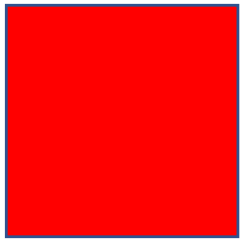


Quick query:

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Why is the int check inside the try?

Exception Handling



Quick query:

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Why is the int check inside the try?

Exception Handling

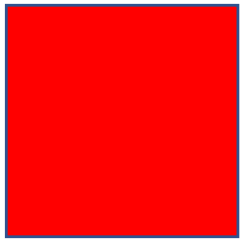
Quick query:

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Why is the int check inside the try?

If it was at the input, then it is outside the try and the cast may fail

Exception Handling



Quick query:

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

Why is the int check inside the try?

If it was at the input, then it is outside the try and the cast may fail

and it is this failing (exception) we are trying to catch

Exception Handling

Quick query:

```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

If I place everything inside the try

Exception Handling

Quick query:

```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

If I place everything inside the try

then run the code to create an exception (input == “u”)

Exception Handling

Quick query:

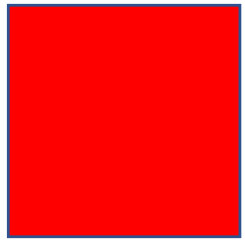
```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

If I place everything inside the try

then run the code to create an exception (input == “u”)

It actually breaks with “number” is not defined

Exception Handling



Quick query:

```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

This is known as “scope”

Exception Handling

Quick query:

```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

This is known as “scope”

Scope states, what ever you define here within the try can only be seen within the try and no where else

Exception Handling

Quick query:

```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("The error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

This is known as “scope”

Scope states, what ever you define here within the try can only be seen within the try and no where else

That is why except can not see it

Exception Handling

Quick query:

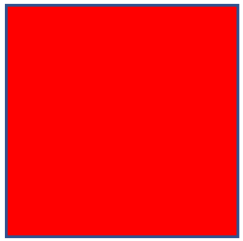
```
try:
    number = int(input("Please enter a positive number: "))
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")

# output
# NameError: name 'number' is not defined
```

We will come across “scope” in lots of places

Mainly in functions and try/except calls

For now know it exists and you might see this kind of problem in your code

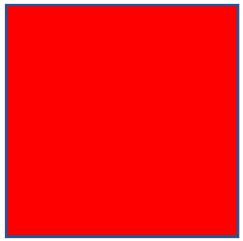
Exception Handling



Let's go back to the original code

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

Exception Handling

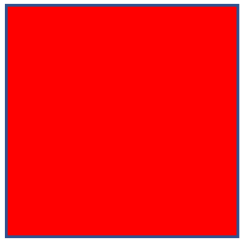


Let’s go back to the original code, and let’s look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

```
Please enter a positive number: 8
8 is even
```

Exception Handling

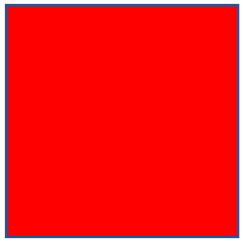


Let’s go back to the original code, and let’s look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

```
Please enter a positive number: 5
5 is odd
```

Exception Handling

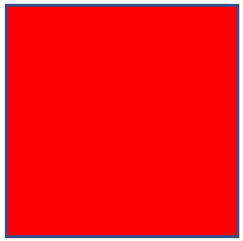


Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

```
Please enter a positive number: -3
-3 is negative, please try again with a positive number
```

Exception Handling



Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

```
Please enter a positive number: h
h is not a number, please retry with a positive number
```

Exception Handling

Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

While I am printing out the number that was entered incorrectly

Sometimes it is nice to see what the actual error was

Exception Handling

Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
```

While I am printing out the number that was entered incorrectly

Sometimes it is nice to see what the actual error was

For this we can print the exception "e"

Exception Handling

Let's go back to the original code, and let's look at some output

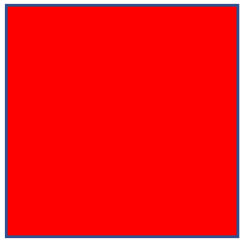
```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

While I am printing out the number that was entered incorrectly

Sometimes it is nice to see what the actual error was

For this we can print the exception "e"

Exception Handling

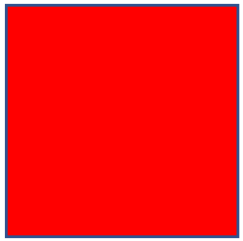


Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

```
Please enter a positive number: h
h is not a number, please retry with a positive number
the error was: "invalid literal for int() with base 10: 'h'"
```

Exception Handling

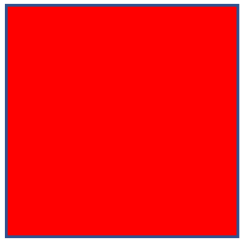


Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

```
Please enter a positive number: 9.0
9.0 is not a number, please retry with a positive number
the error was: "invalid literal for int() with base 10: '9.0'"
```

Exception Handling



Let's go back to the original code, and let's look at some output

```
number = input("Please enter a positive number: ")

try:
    # add back the int check
    number = int(number)
    # check if number is negative
    if number <= 0:
        print(str(number) + " is negative, please try again with a positive number")
    # if positive and even
    elif number % 2 == 0:
        print(str(number) + " is even")
    else:
        print(str(number) + " is odd")
    # if not positive, print error message
except Exception as e:
    print(number + " is not a number, please retry with a positive number")
    print("the error was: \"" + str(e) + "\"")
```

```
Please enter a positive number: (3,6,8)
(3,6,8) is not a number, please retry with a positive number
the error was: "invalid literal for int() with base 10: '(3,6,8)'"
```

Exception Handling Recap

- We introduced if conditional statements over ranges of values
  - num_demodog from 1 to 10
- We added checks to run code when a conditional statement is False

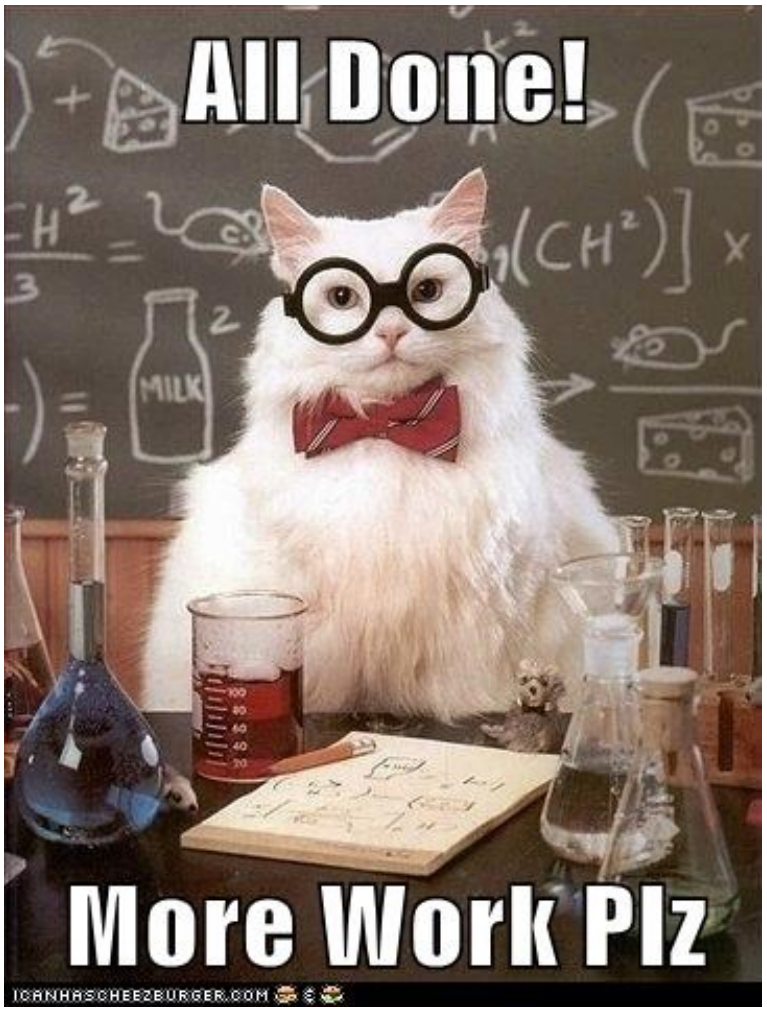
```
if (condition):
    run code if condition is True
else:
    run code if condition is False
```

- And we added checks for multiple inputs using elif

```
if (condition1):
    run code if condition1 is True
elif (condition2):
    run code if condition2 is True
else:
    run code if both condition1 and condition2 are False
```

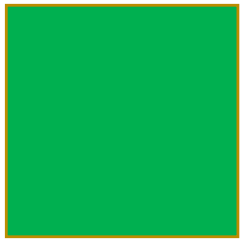
Canvas Student App

Let’s Sign into this lecture now



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Tuple



Tuple

- We've seen Tuple a few times so far in this class
- Tuple creation and assignment:
  - x = ("Ed", "Edd", "Eddy", 2009)

Data Types

Some of the other Data types available in Python

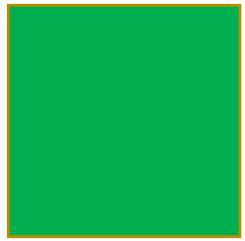
Type	Example
Numeric: Integer, Float	x = 10 x = 10.0
String	x = “Mike”
Boolean	x = True x = False
List	x = [10, 20, 30]
Tuple	x = ("Ed", "Edd", "Eddy", 2009)
Dictionary	x = {‘one’: 1, ‘two’: 2}
List	x = ["Ed", "Edd", "Eddy", 2009]

Tuple

- We've seen Tuple a few times so far in this class
- Tuple creation and assignment:
  - x = ("Ed", "Edd", "Eddy", 2009)

- Functions that return more than one value:
  - print(average_and_modulus_of_two(2,4))
  - # output => (3,0)

Python Functions



It's a Tuple - Mind blown ☺

```
def average_and_modulus_of_two(number_1, number_2):
    ''' this is a 'docstring'
    function to determine the average and modulus of two numbers
    '''
    # determine the average of two numbers
    average = (number_1+number_2) // 2
    # determine the modulus of two numbers
    modulus = (number_1+number_2) % 2
    # return the average and modulus of the two input numbers
    return average, modulus


number_one = 2
number_two = 4
print(average_and_modulus_of_two(number_one, number_two))
# output:
# (3, 0)
```

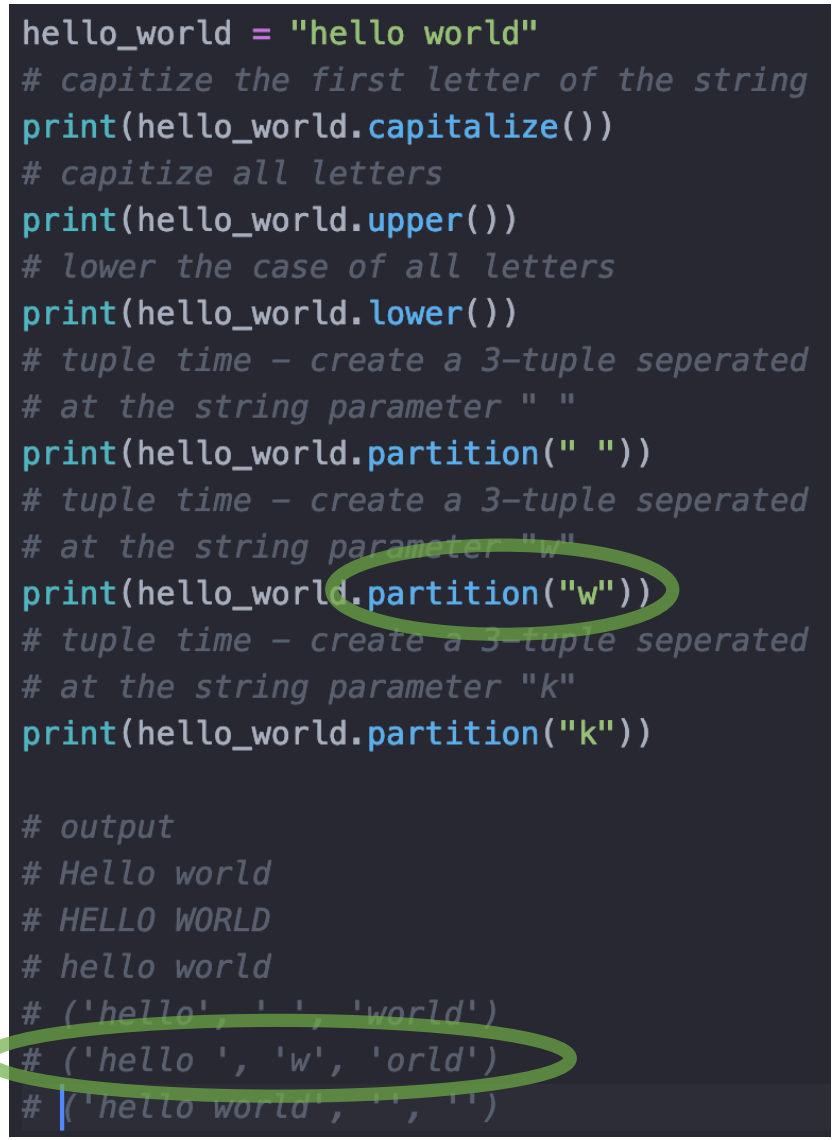
Tuple

- We've seen Tuple a few times so far in this class
- Tuple creation and assignment:
  - x = ("Ed", "Edd", "Eddy", 2009)

- Functions that return more than one value:
  - print(average_and_modulus_of_two(2,4))
  - # output => (3,0)

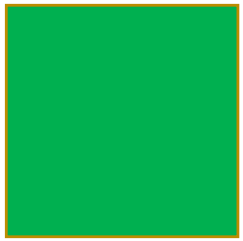
- Using the string partition function:
  - print("hello world".partition("w"))
  - # output => ('hello ', 'w', 'orld')

String Functions



```
hello_world = "hello world"
# capitize the first letter of the string
print(hello_world.capitalize())
# capitize all letters
print(hello_world.upper())
# lower the case of all letters
print(hello_world.lower())
# tuple time - create a 3-tuple seperated
# at the string parameter " "
print(hello_world.partition(" "))
# tuple time - create a 3-tuple seperated
# at the string parameter "w"
print(hello_world.partition("w"))
# tuple time - create a 3-tuple seperated
# at the string parameter "k"
print(hello_world.partition("k"))

# output
# Hello world
# HELLO WORLD
# hello world
# ('hello', ' ', 'world')
# ('hello ', 'w', 'orld')
# ('hello world', '', '')
```

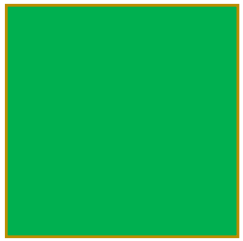
Tuple



As previously stated:

A tuple is an unordered collection of values

Tuple

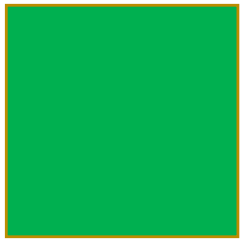


As previously stated:

A tuple is an unordered collection of values

A tuple is immutable (contents cannot be changed)

Tuple



As previously stated:

A tuple is an unordered collection of values

A tuple is immutable (contents cannot be changed)

A tuple is created using ”round brackets”

```
x = ("Ed", "Edd", "Eddy", 2009)
```

Tuple

As previously stated:

A tuple is an unordered collection of values

A tuple is immutable (contents cannot be changed)

A tuple is created using ”round brackets”

x = ("Ed", "Edd", "Eddy", 2009)

A tuple can contain a mixture of variable types

e.g., int, float, string, etc.

Tuple

Let’s look at some output

Assign and print a tuple – option 1

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

# output
# ('Ed', 'Edd', 'Eddy', 2009)
```

Tuple

Let’s look at some output

Assign and print a tuple – option 1

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

# output
# ('Ed', 'Edd', 'Eddy', 2009)
```

The contents of the tuple are printed with round brackets

Tuple

Let’s look at some output

Assign and print a tuple – option 2

```
great_show = tuple(("Ed", "Edd", "Eddy", 2009))
print(great_show)

# output
# ('Ed', 'Edd', 'Eddy', 2009)
```

Tuple

Let’s look at some output

Assign and print a tuple – option 2

```
great_show = tuple(("Ed", "Edd", "Eddy", 2009))
print(great_show)

# output
# ('Ed', 'Edd', 'Eddy', 2009)
```

We can use the tuple() function

Tuple

Let’s look at some output

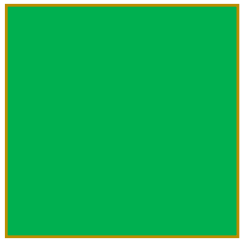
Assign and print a tuple – option 2

```
great_show = tuple(("Ed", "Edd", "Eddy", 2009))
print(great_show)

# output
# ('Ed', 'Edd', 'Eddy', 2009)
```

tuple() takes one argument – which is a tuple

So we must use double brackets

Tuple



Let’s look at some output

We can print individual values in the tuple

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

# output
# Edd
```

Tuple

Let’s look at some output

We can print individual values in the tuple

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

# output
# Edd
```

Using square brackets

Tuple

Let’s look at some output

We can print individual values in the tuple

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

# output
# Edd
```

Using square brackets

We can select the index of the item we want to print

Tuple

Let’s look at some output

We can print individual values in the tuple

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

# output
# Edd
```

We select index 1

“Edd” is printed….

Tuple

Let’s look at some output

We can print individual values in the tuple

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

# output
# Edd
```

We select index 1

“Edd” is printed….

Why?

Tuple

Let’s look at some output

We can print individual values in the tuple

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

# output
# Edd
```

We select index 1

“Edd” is printed….

Remember 0 indexing in CS

Tuple

Let’s look at some output

We can also get the number of items in the tuple

```
great_show = ("Ed", "Edd", "Eddy", 2009)
print("There are "+str(len(great_show))+" items in this tuple")

# output
# There are 4 items in this tuple
```

Tuple

Let’s look at some output

We can also get the number of items in the tuple

```
great_show = ("Ed", "Edd", "Eddy", 2009)
print("There are "+str(len(great_show))+" items in this tuple")

# output
# There are 4 items in this tuple
```

Using the len() function

We can get the number of items in the tuple

Tuple

Let’s look at some output

We can also get the number of items in the tuple

```
great_show = ("Ed", "Ed", "Ed", 2009)
print("There are "+str(len(great_show))+" items in this tuple")

# output
# There are 4 items in this tuple
```

if we change three of the items to the same value

Tuple

Let’s look at some output

We can also get the number of items in the tuple

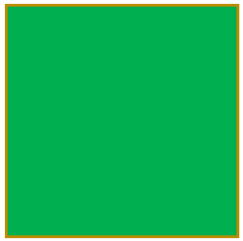
```
great_show = ("Ed", "Ed", "Ed", 2009)
print("There are "+str(len(great_show))+" items in this tuple")

# output
# There are 4 items in this tuple
```

if we change three of the items to the same value

We still get the same result

len() just returns the quantity

Tuple



Let’s look at some output

if we add a new item to the tuple

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

# output
# TypeError: 'tuple' object does not support item assignment
```

Tuple

Let’s look at some output

if we add a new item to the tuple

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

# output
# TypeError: 'tuple' object does not support item assignment
```

Using square brackets, we add “plank” to index 4

Tuple

Let’s look at some output

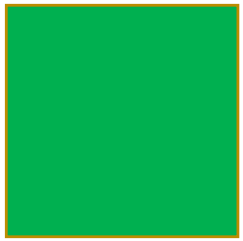
if we add a new item to the tuple

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

# output
# TypeError: 'tuple' object does not support item assignment
```

Using square brackets, we add “plank” to index 4

We get an error – tuple does not support assignment

Tuple



Tuple has two other commonly used functions

count()

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

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

count() returns the number of times
a value appears in a tuple

Tuple

Tuple has two other commonly used functions

count()

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

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

Here we get the value of the first index in great_show

Tuple

Tuple has two other commonly used functions

count()

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

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

Here we get the value of the first index in great_show

Tuple

Tuple has two other commonly used functions

count()

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

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

And count how many times that appears in great_show

Tuple

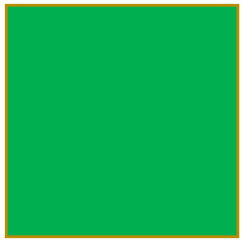
Tuple has two other commonly used functions

count()

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

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

And count how many times that appears in great_show

Tuple



Tuple has two other commonly used functions

index()

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

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

index() returns the first index position that a value appears in a tuple

Tuple

Tuple has two other commonly used functions

index()

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

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

Where does “Ed” appear in great_show

Tuple

Tuple has two other commonly used functions

index()

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

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

Where does “Ed” appear in great_show

Tuple

Let’s look at some output

What happens when we check for a value not in the list

```
great_show = ("Ed", "Ed", "Ed", 2009)
print("ed occurs at index "
      + str(great_show.index("ed"))
      + " in this tuple")

# output
# ValueError: tuple.index(x): x not in tuple
```

Remember “Ed” and “ed” are two different values

Tuple

Let’s look at some output

What happens when we check for a value not in the list

```
great_show = ("Ed", "Ed", "Ed", 2009)
print("ed occurs at index "
      + str(great_show.index("ed"))
      + " in this tuple")

# output
# ValueError: tuple.index(x): x not in tuple
```

We now get an error - ValueError

Tuple

Let's look at some output

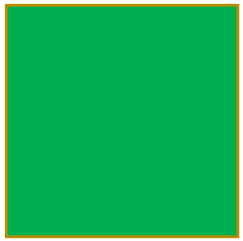
What happens when we check for a value not in the list

```
great_show = ("Ed", "Ed", "Ed", 2009)
print("ed occurs at index "
      + str(great_show.index("ed"))
      + " in this tuple")

# output
# ValueError: tuple.index(x): x not in tuple
```

We now get an error – ValueError

This is no good to us, as this stops the program, so…

Tuple



Let’s look at some output

Let’s add some exception handling - a try/except

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

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

Tuple

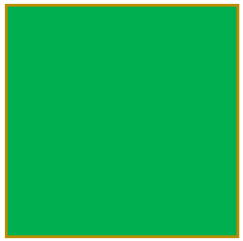
Let’s look at some output

Let’s add some exception handling - a try/except

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

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

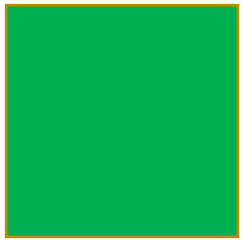
Now we get a print statement telling us there is a problem

Tuple recap



We can now create tuples – 2 different ways

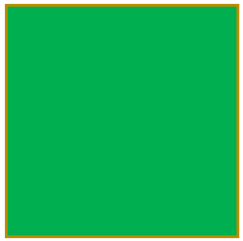
Tuple recap



We can now create tuples – 2 different ways

We can get a value based on index number

Tuple recap

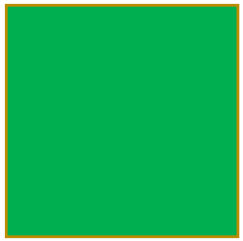


We can now create tuples – 2 different ways

We can get a value based on index number

We can get the length of a tuple

Tuple recap



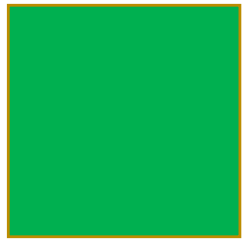
We can now create tuples – 2 different ways

We can get a value based on index number

We can get the length of a tuple

We can't add to a tuple – immutable

Tuple recap



We can now create tuples – 2 different ways

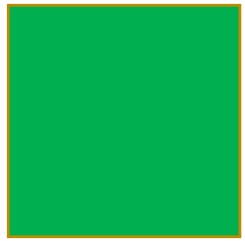
We can get a value based on index number

We can get the length of a tuple

We can't add to a tuple – immutable

We can count how many times a value appears in a tuple

Tuple recap



We can now create tuples – 2 different ways

We can get a value based on index number

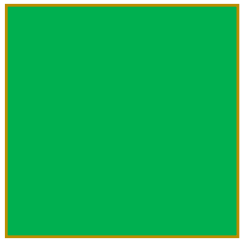
We can get the length of a tuple

We can't add to a tuple – immutable

We can count how many times a value appears in a tuple

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

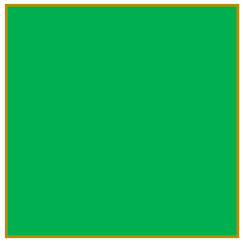
Tuple



So after 40 slides on tuples, I get to the point I want to make

Because we can now determine if a value is in a tuple

Tuple



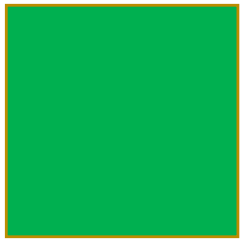
So after 40 slides on tuples, I get to the point I want to make

Because we can now determine if a value is in a tuple

We can write

```
if value in tuple:
```

Tuple



So after 40 slides on tuples, I get to the point I want to make

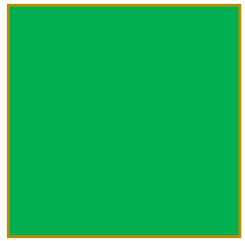
Because we can now determine if a value is in a tuple

We can write

if value in tuple:

And this will return a True or False

Tuple



So after 40 slides on tuples, I get to the point I want to make

Because we can now determine if a value is in a tuple

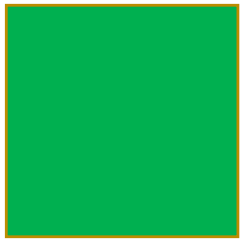
We can write

if value in tuple:

And this will return a True or False

This is known as membership

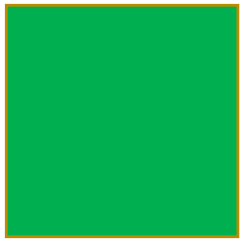
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

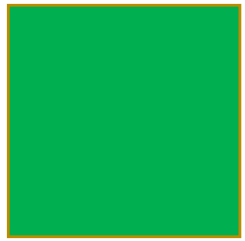
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

Tuple



if value in tuple:

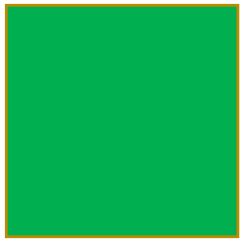
```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

Tuple

if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

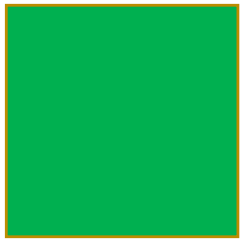
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

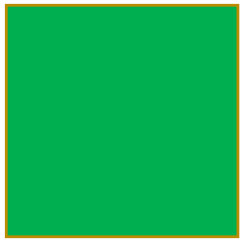
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

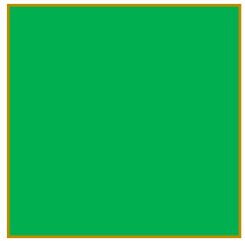
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

# output
# ed does not occurs in ('Ed', 'Ed', 'Ed', 2009)
```

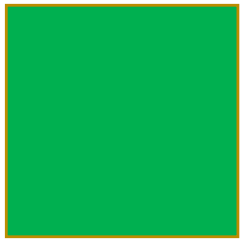
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "Ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

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

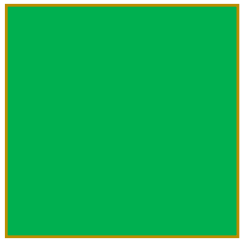
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "Ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

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

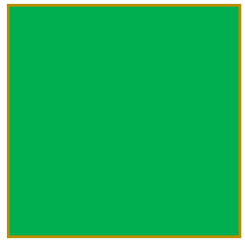
Tuple



if value in tuple:

```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "Ed"

if value_to_find in great_show:
    print(value_to_find + " occurs at index "
          + str(great_show.index(value_to_find))
          + " in this tuple")
else:
    print(value_to_find + " does not occurs in "
          + str(great_show))

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

Tuple recap



if value in tuple:

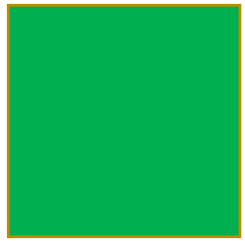
So we can use if to find out if a value is in a tuple

And we do not need to use try/except

This you can use for if/elif/else

Cool ☺

Tuple recap



Oh and if we can use

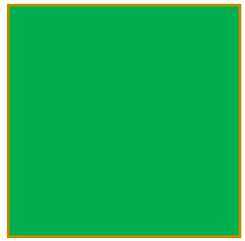
```
if value in tuple:
```

We can also use:

```
if value not in tuple:
```

To negate an input, without using the else

Tuple recap



```
great_show = ("Ed", "Ed", "Ed", 2009)
value_to_find = "ed"

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)
```

To negate an input, without using the else

Live Coding Time…

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh