Lecture 6: Errors and Exceptions

Contents today

1. Recap:
    - Data structures
    - Control flow (similar to and relevant for today's content)
    - Functions
2. Why do errors even exist?
3. Receiving Errors and dealing with them

Contents today

1. Recap:
    - Data structures
    - Control flow (similar to and relevant for today's content)
    - Functions
2. Why do errors even exist?
3. Receiving Errors and dealing with them

1. You're programming now! Return the favour and serve the user some Errors

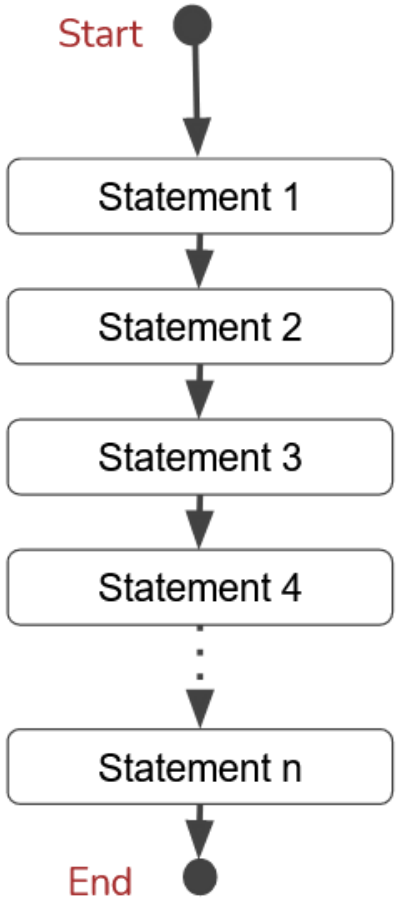
Recap: Data structures

	Changeable	Duplicates	Order	Indexed
Tuples	No	Yes	Yes	Yes
Sets	Yes	No	No	No
Dictionaries	Yes	No (ish)	No	Yes (by key)
Lists	Yes	Yes	Yes	Yes

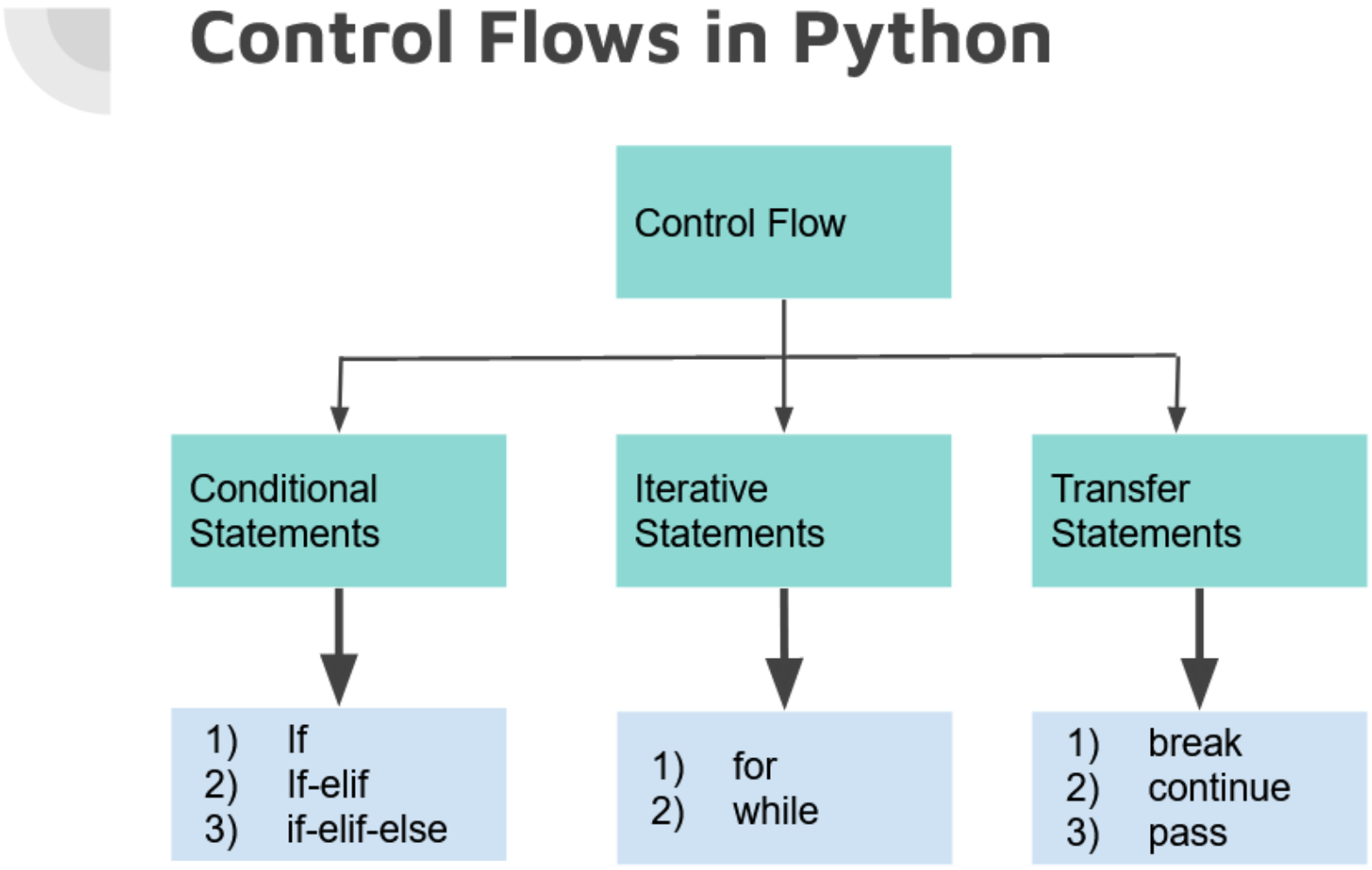
Recap: Control Flow

Program Statements

- We said that to create programs we need to instruct the computer what to do
  - We instruct by creating programing statements
  - The computer can run them sequentially to achieve an output
  - Sometimes however you might need to repeat a statement or
  - Only run a statement when some conditions hold



Recap: Control Flow



Recap: Control Flow

For loops

A for loop is used for iterating over a sequence

- list, tuple, dictionary, set, or string

Example

```
for letter in 'Python':
    print('Current Letter :', letter)
```

Useful to remember: a string is also a sequence!

Useful to remember: a string is also a sequence!

In [1]:
```
for letter in 'Python':
    print(letter)
```

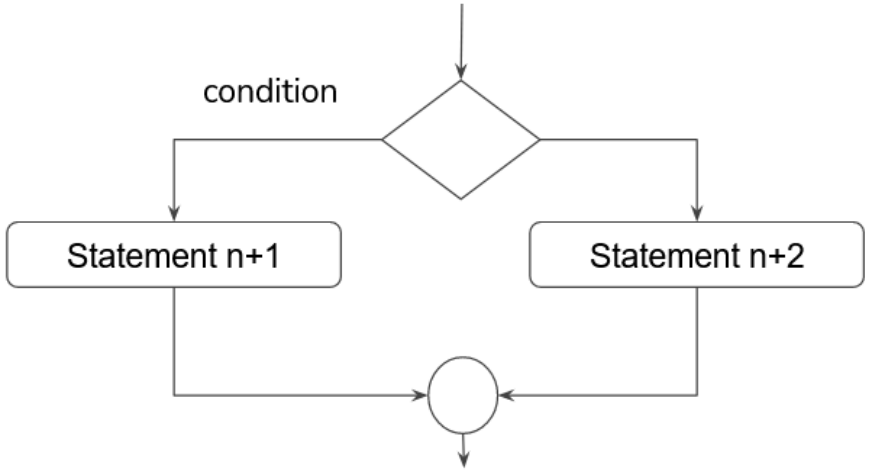
```
P
y
t
h
o
n
```

Recap: Control Flow

Conditional Statements

If statements were shown in the previous examples. With an if statement you place one or more conditions on the execution of following statements

```
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```



Functions

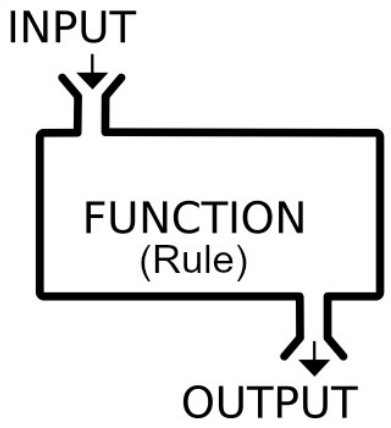
You have seen methods as part of objects, for example the `__init__` method or the `getPatientCount()` method on the patient object. Methods take parameters as input (id, name, age etc) and **do** something with/to them. Often they also `return` something (some 'output').

Methods are defined by using the `def` keyword.

```
class Patient:

    def __init__(self, name):
        self.name=name

    def getPatientCount():
        return Patient.count
```



A **function** is very similar to methods. The difference being that a function is not part of an object like a method is. You might have seen and used functions already, for example:

Functions

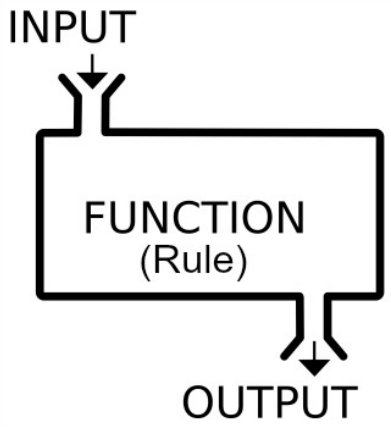
You have seen methods as part of objects, for example the `__init__` method or the `getPatientCount()` method on the patient object. Methods take parameters as input (id, name, age etc) and **do** something with/to them. Often they also `return` something (some 'output').

Methods are defined by using the `def` keyword.

```
class Patient:

    def __init__(self, name):
        self.name=name

    def getPatientCount():
        return Patient.count
```



A **function** is very similar to methods. The difference being that a function is not part of an object like a method is. You might have seen and used functions already, for example:

In [2]:

```
list_of_bmis = [22, 25, 18, 29]
max(list_of_bmis)
```

Out[2]:

```
29
```

You can define your own functions by using the `def` keyword

You can define your own functions by using the `def` keyword

In [3]:

```
def bmi_classification(bmi):
    if bmi < 20:
        return 'underweight'
    elif bmi < 25:
        return 'normal weight'
    else:
        return 'overweight'
```

You can then use this function in your own programs

You can then use this function in your own programs

In [4]:
```
bmi_classification(24.5)
```

Out[4]:
```
'normal weight'
```

You can then use this function in your own programs

In [4]:
```
bmi_classification(24.5)
```

Out[4]:
```
'normal weight'
```

In [5]:
```
list_of_bmis = [22, 25, 18, 29]

for bmi in list_of_bmis:
    print(bmi_classification(bmi))
```

```
normal weight
overweight
underweight
overweight
```

You already learned a lot!

- Data types
    - Numbers/Integer/Float/String/Boolean/Sequence
- Data structures
    - List/Dictionary/Set/Tuple
- Variables
- Object Oriented Programming
    - Classes/Objects/Attributes/Methods
- Operators
    - Arithmetic/Assignment/Comparison/Logical/Identity/Membership
- Control Flow
    - If/elif/else
    - for/while
    - pass/break/continue

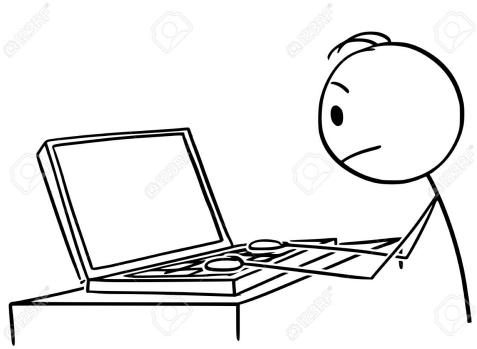
A lot of it by trial and error...

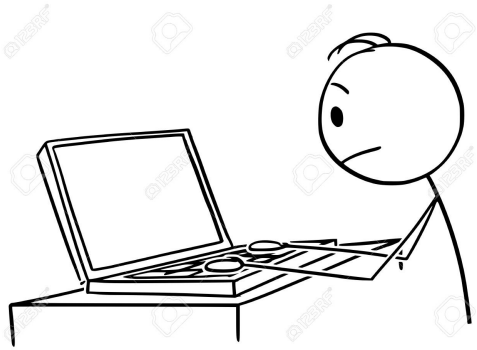
"Create a list of 4 names and access the fourth name in the list"

```
four names = ['Visara', 'Vikas', 'Sil', Niels]
```

```
  File "<ipython-input-5-e3992080879a>", line 1
    four names = ['Visara', 'Vikas', 'Sil', Niels]
               ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW



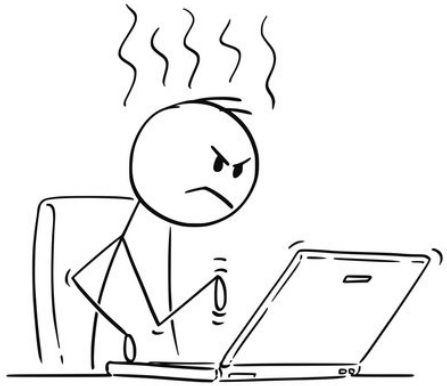


```
four_names = ['Visara', 'Vikas', 'Sil', Niels]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-3e8c2f32733f> in <module>()
----> 1 four_names = ['Visara', 'Vikas', 'Sil', Niels]

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

SEARCH STACK OVERFLOW

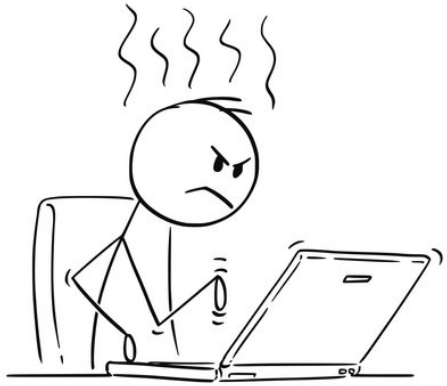


```
four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
four_names[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-f9f35648ebc1> in <module>()
      1 four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
----> 2 four_names[4]

IndexError: list index out of range
```

SEARCH STACK OVERFLOW



```
four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
four_names[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-f9f35648ebc1> in <module>()
      1 four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
----> 2 four_names[4]

IndexError: list index out of range
```

SEARCH STACK OVERFLOW


DON'T
PANIC!

Errors are not as problematic as they appear to be

Errors are not as problematic as they appear to be

They are just trying to tell you something, so let's have a closer look

1. Errors and Exceptions: Why do they even exist?

1. Errors and Exceptions: Why do they even exist?

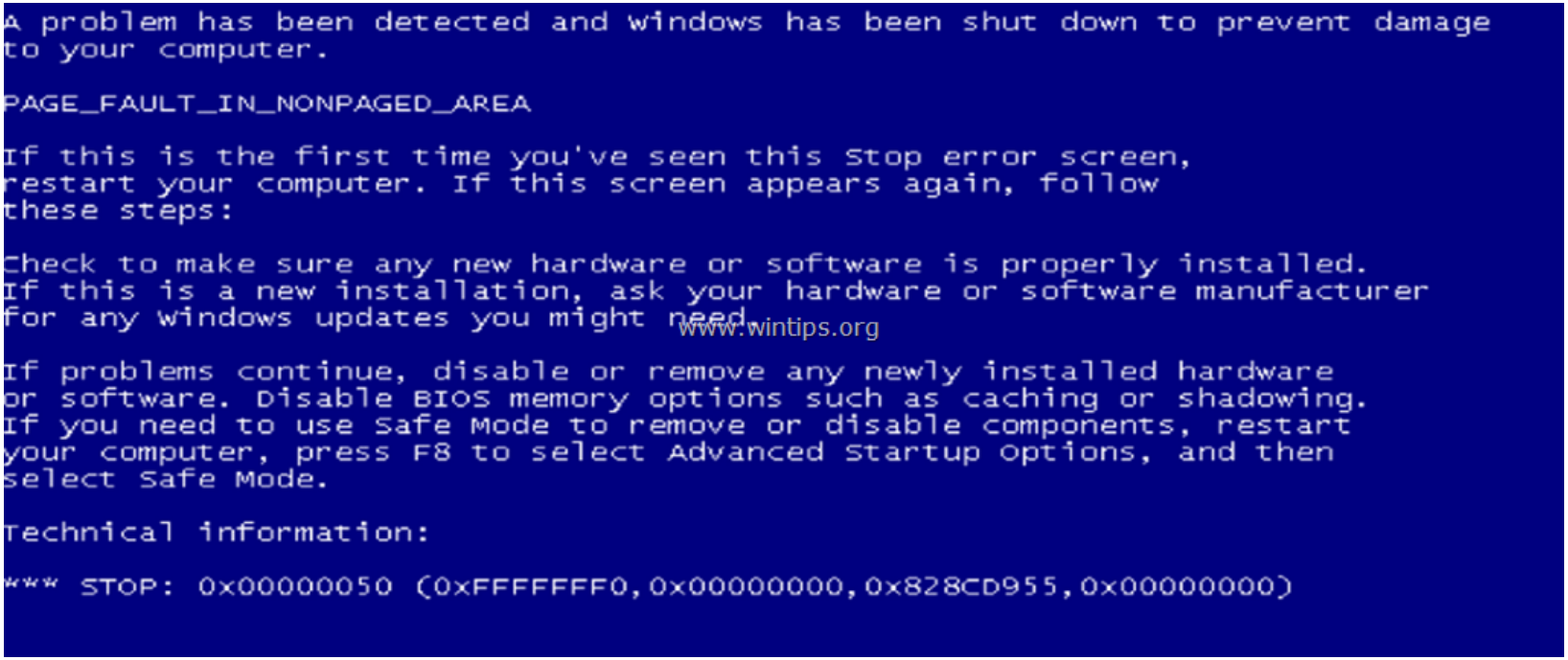
Well. Because things go wrong.

1. Errors and Exceptions: Why do they even exist?

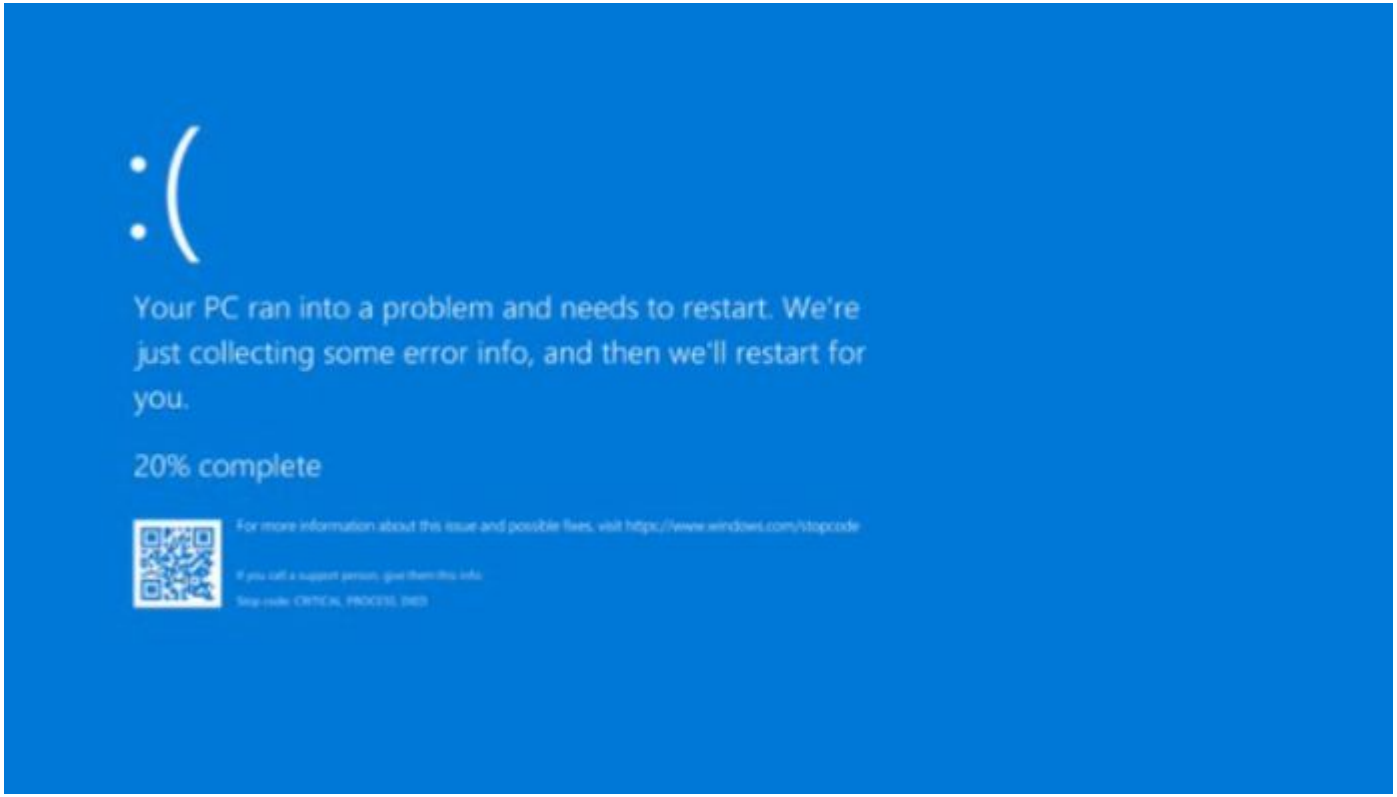
Well. Because things go wrong.

And when they go wrong, we don't want everything to come crashing down...

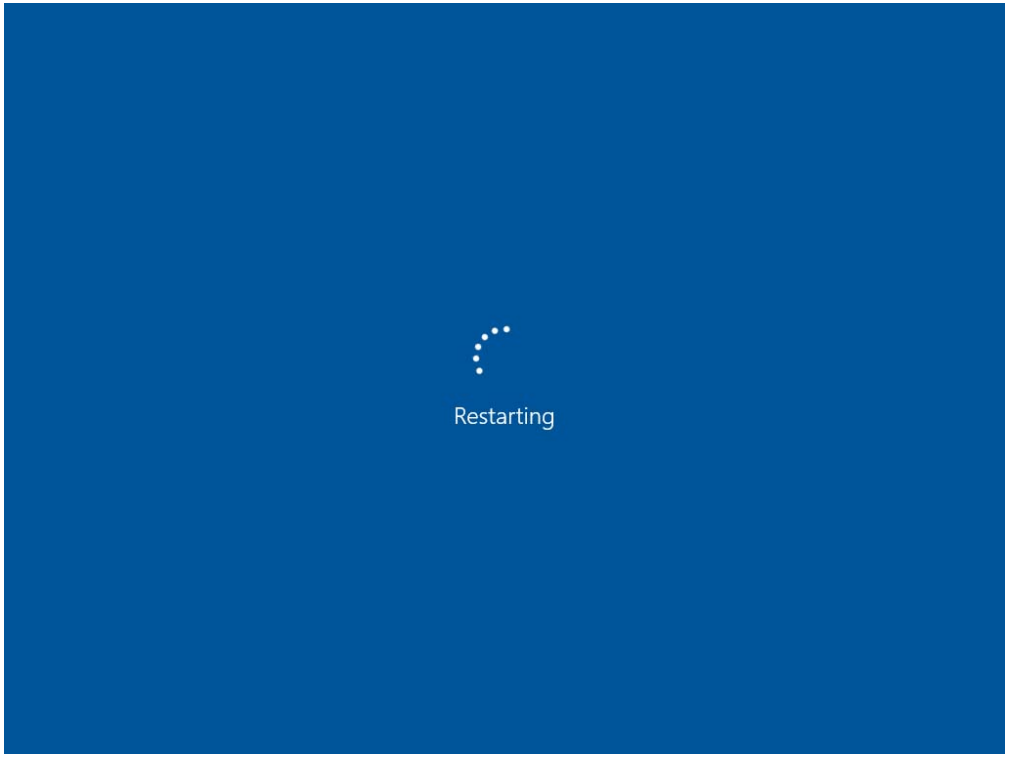
Anyone remember this?



Probably you guys are more familiar with

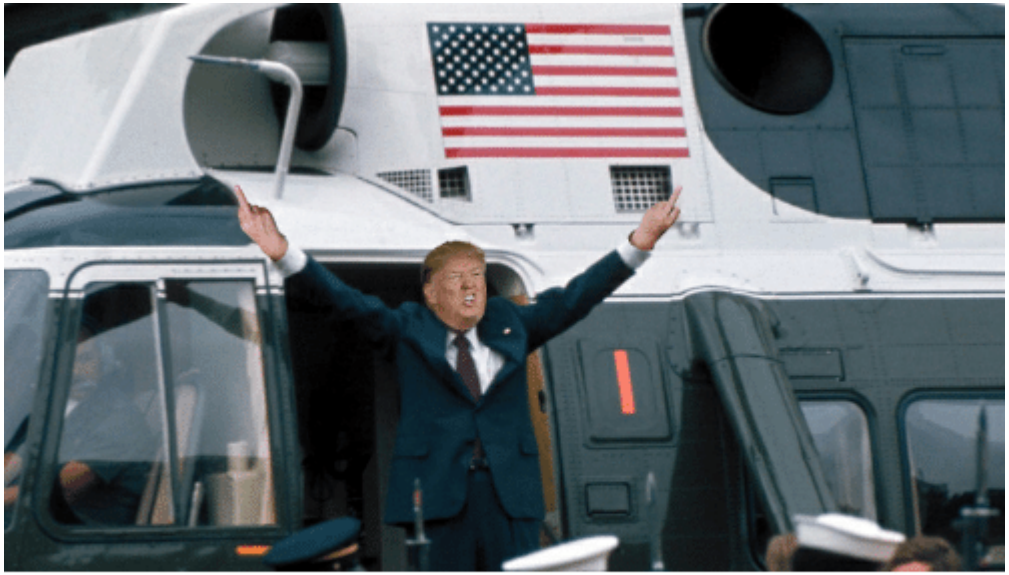


But either way, restarting the entire computer is not a very elegant way to handle something going wrong in a program



This is their purpose!

Errors help the program exit gracefully when something goes (horribly) wrong



Time for a graceful exit.

In [41]:

```
class Patient:

    def __init__(self, name)
        self.name=name
```

```
  File "C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/2779481092.py", line 3
    def __init__(self, name)
                            ^
SyntaxError: invalid syntax
```

In [41]:
```
class Patient:

    def __init__(self, name)
        self.name=name
```

```
  File "C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/2779481092.p
y", line 3
    def __init__(self, name)
                            ^
SyntaxError: invalid syntax
```

It's not pleasant to look at, but it is a whole lot better than restarting the computer

TODO?

Add a little history to illustrate how errors were dealt with in the past (and still in C and Fortran). Status codes. Lot of cluttered code.

as here from #option 1 onwards with the green and red code.

Should illustrate the point that it is important to separate what's supposed to happen and what can go wrong.

But might become too much.

2. Errors and Exceptions: Receiving and dealing with them

There are 2 broad classes, namely:

1. Syntax Errors: There's something wrong with the 'grammar' of our code
2. Exceptions: Errors detected during execution are called exceptions and are not unconditionally fatal

If an exception is not *handled*, it will result in an error message

Components of an error message

- Error type
- Traceback
- There's arrows!

Components of an error message

- Error type
- Traceback
- There's arrows!

In [25]:

```
class Patient:

    def __init__(self, name):
        self.name=name
        self.ethnicity=ethnicity

p1 = Patient('Niels')
```

```
---------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/4112194549.py in <mo
dule>
      5         self.ethnicity=ethnicity
      6
----> 7 p1 = Patient('Niels')

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/4112194549.py in __i
nit__(self, name)
      3     def __init__(self, name):
      4         self.name=name
----> 5         self.ethnicity=ethnicity
      6
      7 p1 = Patient('Niels')

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

Types of errors

- SyntaxError
- NameError
- IndexError
- IndentationError
- KeyError
- FileNotFoundError
- IOError
- ImportError
- TypeError
- UnicodeError
- ValueError
- ZeroDivisionError

Let’s look at a few of these

SyntaxError

There's something wrong with the 'grammar' of our code

For example, python expects a method definition to end with a colon `:`

SyntaxError

There's something wrong with the 'grammar' of our code

For example, python expects a method definition to end with a colon :

In [26]:
```
class Patient:

    def __init__(self, name)
        self.name=name
```

```
  File "C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/2779481092.py", line 3
    def __init__(self, name)
                            ^
SyntaxError: invalid syntax
```

NameError

Usually we are trying to use/access a variable before it has been defined. Before we assigned a value to it

NameError

Usually we are trying to use/access a variable before it has been defined. Before we assigned a value to it

In [27]:
```
a = 1
a
```

Out[27]:
```
1
```

NameError

Usually we are trying to use/access a variable before it has been defined. Before we assigned a value to it

In [27]:
```
a = 1
a
```

Out[27]:
```
1
```

In [28]:
```
b
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/1685013873.py in <module>
----> 1 b

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

IndentationError

IndentationError

In [30]:

```
a = 2
b = 1

if a > b:
    print('comparing...')
 print('a is larger than b')
```

```
  File "<tokenize>", line 6
    print('a is larger than b')
    ^
IndentationError: unindent does not match any outer indentation level
```

KeyError

If a key value cannot be found in a data structure, such as a dictionary

KeyError

If a key value cannot be found in a data structure, such as a dictionary

In [31]:
```
students = {1: 'Anton', 2: 'Bettina', 3: 'Marijn'}
students[5]
```

```
---------------------------------------------------------------------
-
KeyError                                  Traceback (most recent call las
t)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/169312155.py in <mod
ule>
      1 students = {1: 'Anton', 2: 'Bettina', 3: 'Marijn'}
----> 2 students[5]

KeyError: 5
```

ImportError/ModuleNotFoundError

Until now we've only been using python modules which are installed by default (the standard library), such as `math`, but later you will also be using additional modules which are not part of the standard library and need to be installed first.

ImportError/ModuleNotFoundError

Until now we've only been using python modules which are installed by default (the standard library), such as `math`, but later you will also be using additional modules which are not part of the standard library and need to be installed first.

In [32]:
```
import tensorflow
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/4294963926.py in <module>
----> 1 import tensorflow

ModuleNotFoundError: No module named 'tensorflow'
```

StackOverflow

StackOverflow

```
four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
four_names[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-f9f35648ebc1> in <module>()
      1 four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
----> 2 four_names[4]

IndexError: list index out of range
```

SEARCH STACK OVERFLOW

Search StackOverflow

Errors and Exceptions: Handling

So what do we use this knowledge for? What are the benefits?

1. Seperate in your program:
    - what's supposed to happen when everything works, from...
    - ...what happens when things go wrong
2. Prevent our program from crashing if it is used in a non-proper way, by writing code to handle exceptional cases

Let's look at some examples

Try/Except: Example 1

User input

Try/Except: Example 1

User input

Some of you have already explored the input() function, which asks the user to input a value. Using this, the value that is entered is automatically of type string.

Try/Except: Example 1

User input

Some of you have already explored the input() function, which asks the user to input a value. Using this, the value that is entered is automatically of type string.

In [33]:
```
a = input('Enter a value here: ')
type(a)
```

```
Enter a value here:
```

Out[33]:
```
str
```

So if you want the user to enter a number, say an integer, you need to change the data type yourself.

So if you want the user to enter a number, say an integer, you need to change the data type yourself.

In [34]:
```
a = input('Give me an integer: ')

# the int() function turns a value into an integer
int(a)
```

```
Give me an integer:

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/3058163054.py in <module>
      2
      3 # the int() function turns a value into an integer
----> 4 int(a)

ValueError: invalid literal for int() with base 10: ''
```

If the int() function receives a value it cannot turn into an integer, it raises a ValueError. You can use this knowledge to prevent your program from breaking and stopping when a user enters a non-integer, like so:

If the int() function receives a value it cannot turn into an integer, it raises a ValueError. You can use this knowledge to prevent your program from breaking and stopping when a user enters a non-integer, like so:

In [35]:
```
while True:
    try:
        x = int(input('Please enter a number: '))
        break
    except ValueError:
        print('Oops!  That was no valid number.  Try again...')

print('You (finally) entered a valid number and it was: ', x)
```

```
Please enter a number:
Oops!  That was no valid number.  Try again...
Please enter a number:
Oops!  That was no valid number.  Try again...
Please enter a number:
Oops!  That was no valid number.  Try again...
Please enter a number: 10
You (finally) entered a valid number and it was:  10
```

Try/Except: Example 2

File reading

Try/Except: Example 2

File reading

In [36]:
```
# open a file from disk, like this notebook
with open('Lecture06-ErrorsAndExceptions.ipynb', 'r') as f:
    # print the first part
    print(f.read()[:50])
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "
```

In [37]:

```
try:
    # open a file from disk, like this notebook
    with open('Lecture06-ErrorsAndExceptions2.ipynb', 'r') as f:
        # print the first part
        print(f.read()[:40])

except FileNotFoundError as e:
    print('You are probably using the wrong filename. Encountered an error:')
    print(e)
```

```
You are probably using the wrong filename. Encountered an error:
[Errno 2] No such file or directory: 'Lecture06-ErrorsAndExceptions2.ipynb
'
```

3. Errors and Exceptions: Raising

If you want to signal to the user that something went wrong, you can `raise` an error.

For example, the `Patient` class from week 4 could be extended with an updateAge() method, which updates (sets) the age of a patient. In order to prevent strings being set as age, you should do a check to see if the input is really an integer.

One way to do this is by using if/else

One way to do this is by using if/else

In [38]:

```
# Patient Class
class Patient:

    def __init__(self, name, gender, age):
        self.name=name
        self.gender=gender
        self.updateAge(age)


    # method for updating the age of the patient
    def updateAge(self, age):
        if (type(age) == int): # checks that the age is an integer
            self.age=age
        else:
            print("Age was not updated. Age must be a number!")

p1 = Patient('Niels', 'M', '39')
p1.age
```

```
Age was not updated. Age must be a number!

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/3340607542.py in <module>
     16
     17 p1 = Patient('Niels', 'M', '39')
---> 18 p1.age

AttributeError: 'Patient' object has no attribute 'age'
```

One way to do this is by using if/else

In [38]:
```
# Patient Class
class Patient:

    def __init__(self, name, gender, age):
        self.name=name
        self.gender=gender
        self.updateAge(age)


    # method for updating the age of the patient
    def updateAge(self, age):
        if (type(age) == int): # checks that the age is an integer
            self.age=age
        else:
            print("Age was not updated. Age must be a number!")

p1 = Patient('Niels', 'M', '39')
p1.age
```

```
Age was not updated. Age must be a number!

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/3340607542.py in <module>
     16
     17 p1 = Patient('Niels', 'M', '39')
---> 18 p1.age

AttributeError: 'Patient' object has no attribute 'age'
```

Remember though, that it is helpful to the readers of your code if they can **separate what's supposed to happen** from **what's going wrong**.

Even though a message is printed, we could make it clearer that something goes wrong if

age is not an integer.

Moreover, as you can see above, the program is not terminated once age is invalid at line 17, but line 18 is still run.

Another way: `raise a ValueError`

Another way: `raise a ValueError`

In [39]:

```
# Patient Class
class Patient:

    def __init__(self, name, gender, age):
        self.name=name
        self.gender=gender
        self.updateAge(age)


    # method for updating the age of the patient
    def updateAge(self, age):
        if (type(age) == int): # checks that the age is an integer
            self.age=age
        else:
            raise ValueError("Age must be a number! Age was not updated. ")

p1 = Patient('Niels', 'M', '39')
p1.age
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/3830491003.py in <module>
     15             raise ValueError("Age must be a number! Age was not updated. ")
     16
---> 17 p1 = Patient('Niels', 'M', '39')
     18 p1.age

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/3830491003.py in __init__(self, name, gender, age)
      5         self.name=name
      6         self.gender=gender
----> 7         self.updateAge(age)
      8
```

```
      9

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/3830491003.py in upd
ateAge(self, age)
     13             self.age=age
     14         else:
---> 15             raise ValueError("Age must be a number! Age was not up
dated. ")
     16
     17 p1 = Patient('Niels', 'M', '39')

ValueError: Age must be a number! Age was not updated.
```

A more concise way to do something like this, is by using an `assert` statement. We can use this in the following way:

A more concise way to do something like this, is by using an `assert` statement. We can use this in the following way:

In [40]:
```
# Patient Class
class Patient:

    def __init__(self, name, gender, age):
        self.name=name
        self.gender=gender
        self.updateAge(age)


    # method for updating the age of the patient
    def updateAge(self, age):

        assert (type(age) == int), "Age is not an integer" # note: message describes what

        # if the assert statement passes, the code keeps running, otherwise an AssertionE
        self.age=age

p1 = Patient('Niels', 'M', '39')
p1.age
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/801670399.py in <module>
     16         self.age=age
     17
---> 18 p1 = Patient('Niels', 'M', '39')
     19 p1.age

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/801670399.py in __init__(self, name, gender, age)
      5         self.name=name
      6         self.gender=gender
----> 7         self.updateAge(age)
```

```
      8
      9

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_424/801670399.py in upda
teAge(self, age)
     11     def updateAge(self, age):
     12
---> 13         assert (type(age) == int), "Age is not an integer" # note:
message describes what happens when the statement fails
     14
     15         # if the assert statement passes, the code keeps running,
otherwise an AssertionError is raised

AssertionError: Age is not an integer
```

Takeaway messages

- Don't be scared of errors.
- Read them. Google them.
- Errors and Exceptions seperate:
    - what's supposed to happen, from...
    - ...what can go wrong
- Handle exceptions by using `try` and `except`
- `raise` errors to signal to user something went wrong

In [ ]:

In [ ]: