Intro to Python Programming

Lecture 6: Errors and Exceptions

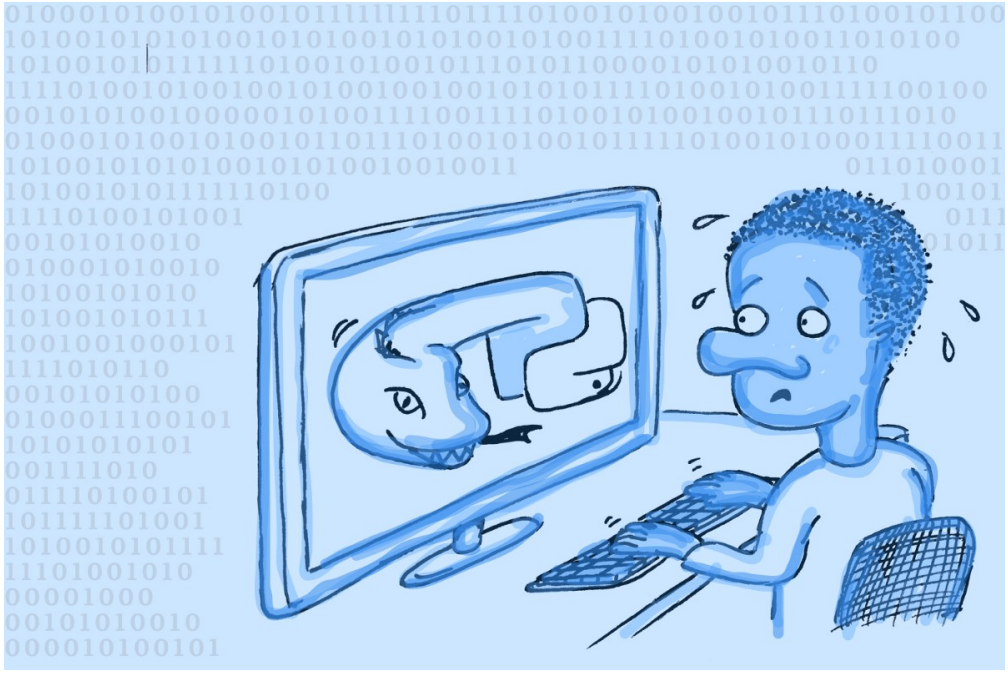


image by Ari Joury

Contents today

1. Recap:
    - Data structures
    - Control flow (today's content is an extension of control flow)
2. Why do errors even exist?
3. Receiving Errors and dealing with them

Contents today

1. Recap:
    - Data structures
    - Control flow (today's content is an extension of control flow)
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

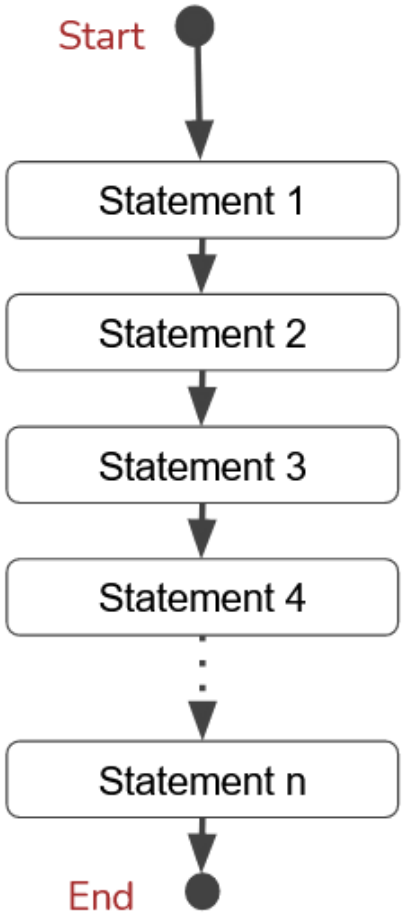
Recap: Data structures

	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

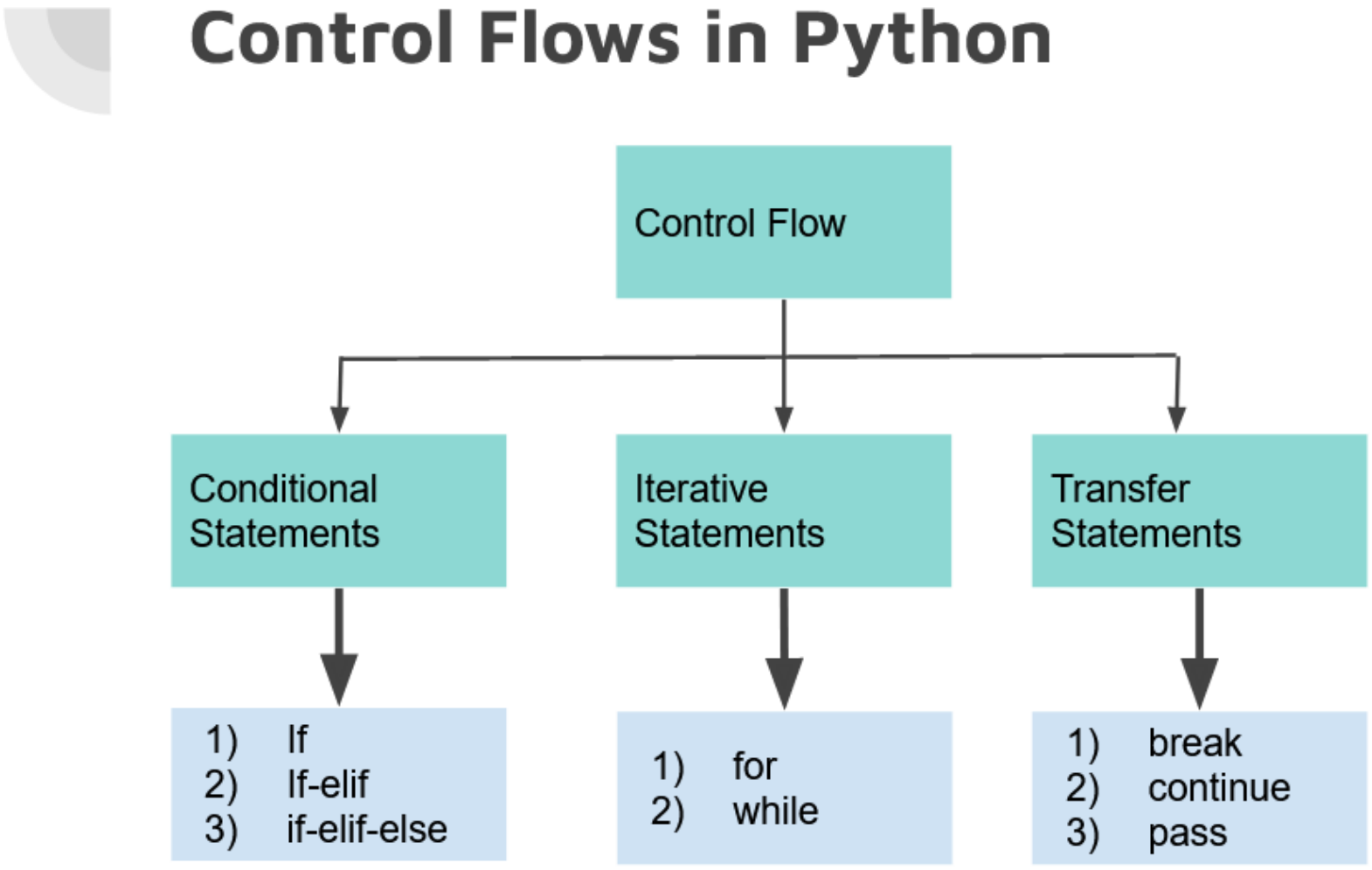
Recap: Control Flow

Program Statements

- We said that to create programs we need to instruct the computer what to do
  - We instruct by creating programming 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)
```

In [1]:

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

# if you simply need to iterate over something, a while loop is NOT the right solution
i = 0
while i < len(four_names):
    print(four_names[i])
    i += 1
```

```
Visara
Vikas
Sil
Niels
```

In [1]:
```
four_names = ['Visara', 'Vikas', 'Sil', 'Niels']

# if you simply need to iterate over something, a while loop is NOT the right solution
i = 0
while i < len(four_names):
    print(four_names[i])
    i += 1
```

```
Visara
Vikas
Sil
Niels
```

In [2]:
```
# use a for loop instead
for name in four_names:
    print(name)
```

```
Visara
Vikas
Sil
Niels
```

You might be saying: "*But the while loop also works!*" Yes it does, but...

You might be saying: "*But the while loop also works!*" Yes it does, but...

In [3]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Useful to remember: a string is also a sequence!

Useful to remember: a string is also a sequence!

In [4]:
```
for letter in 'Python':
    print(letter)
```

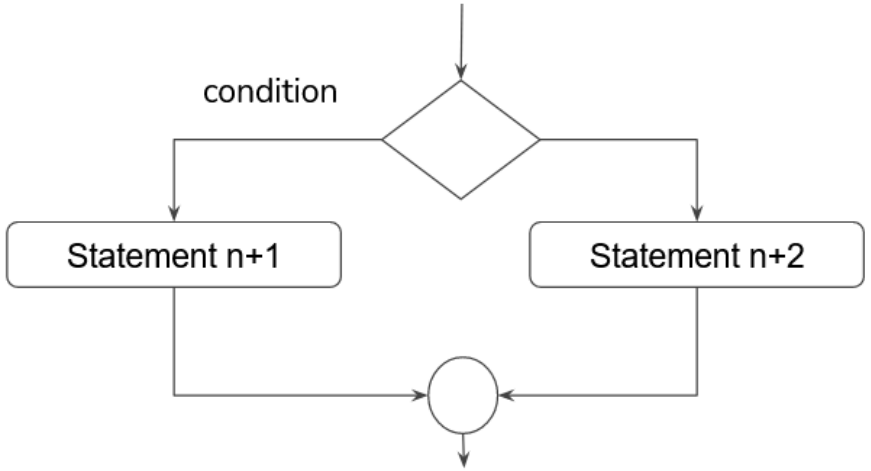
```
P
y
t
h
o
n
```

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")
```



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

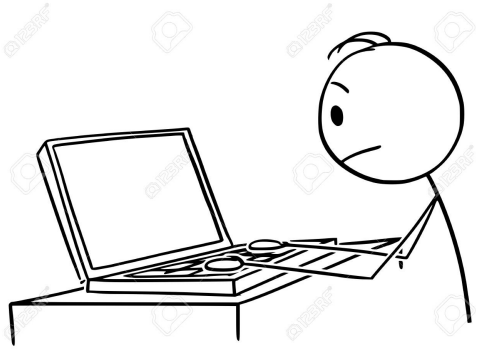
A lot of it by trial and error...

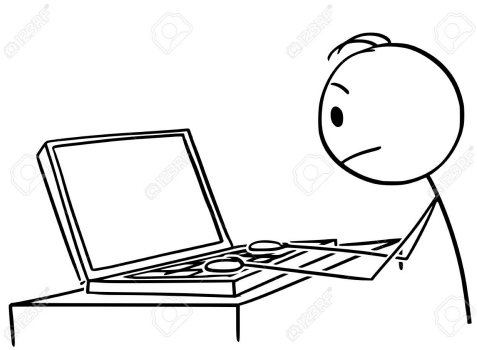
"Create a list of 4 names and access the fourth name in the list"

```
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



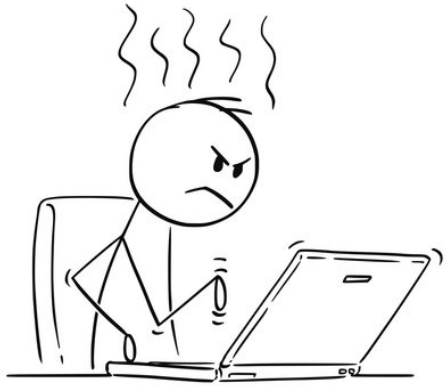


```
four_names = ['Visara', 'Vikas', 'Sil', Niels]
```

```
---------------------------------------------------------------------------
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

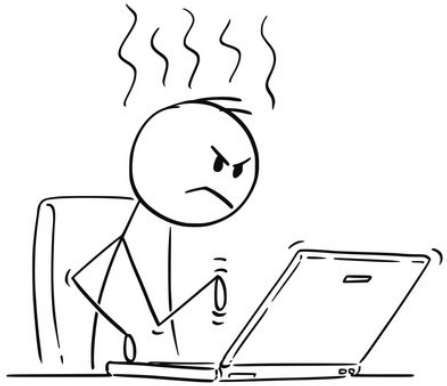


```
four_names = ['Visara', 'Vikas', 'Sil', 'Niels']
four_names[4]
```

```
---------------------------------------------------------------------------
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?

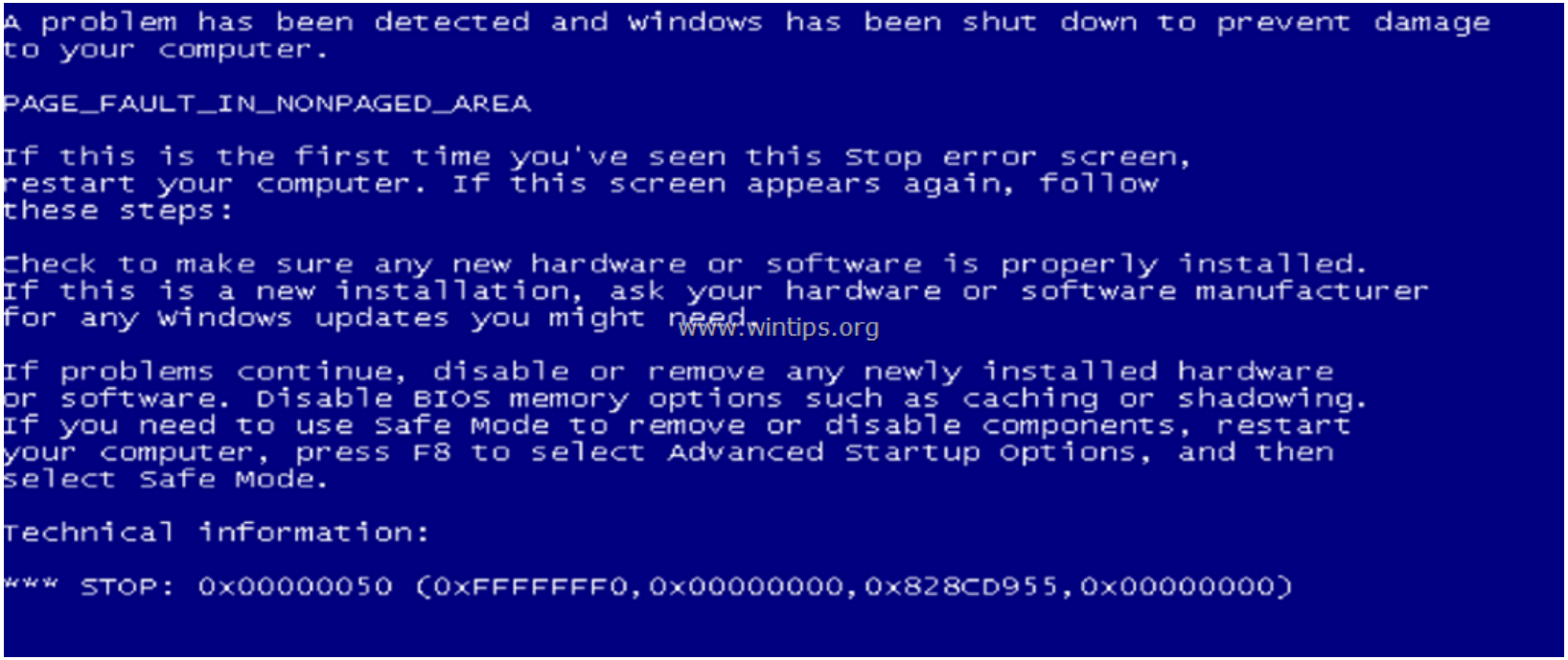
Well. Because things go wrong.

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

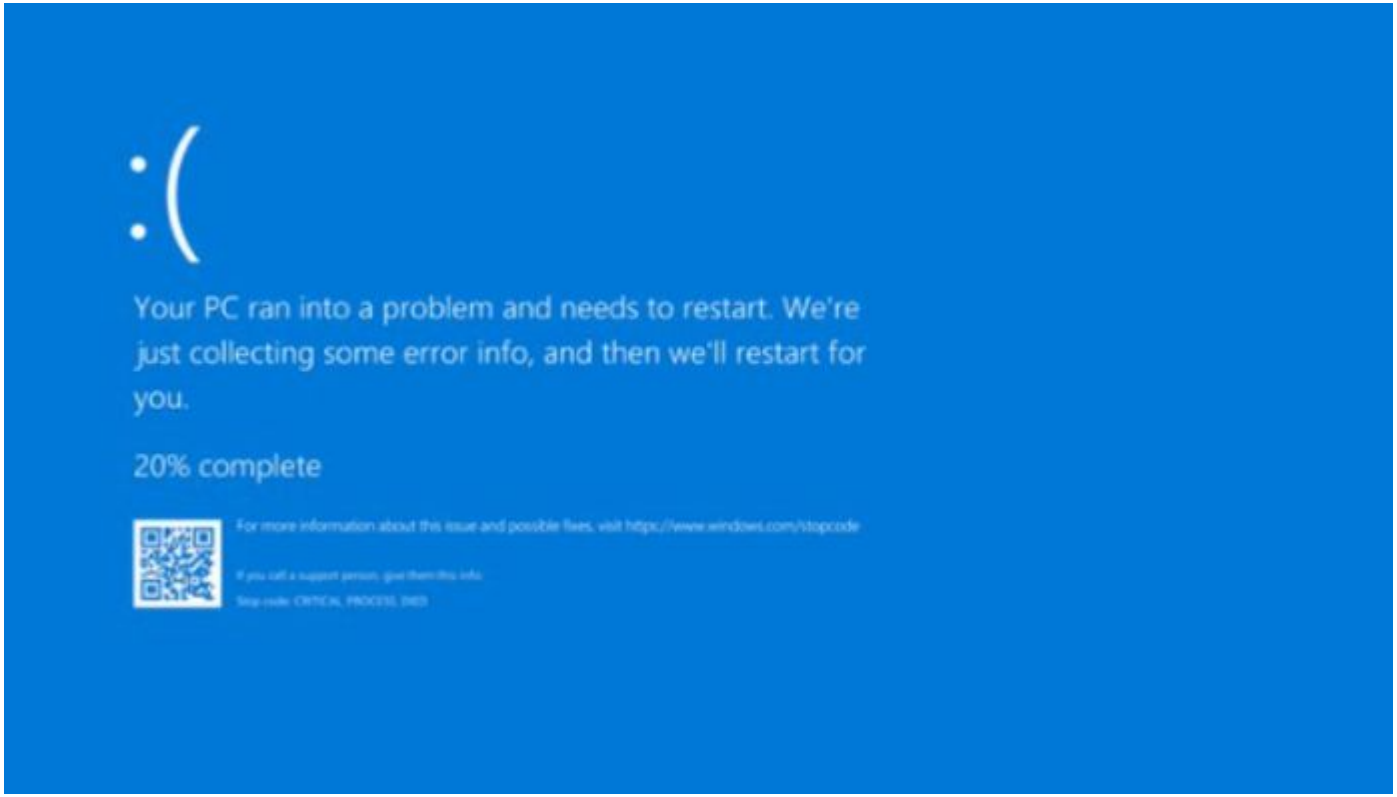
Well. Because things go wrong.

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

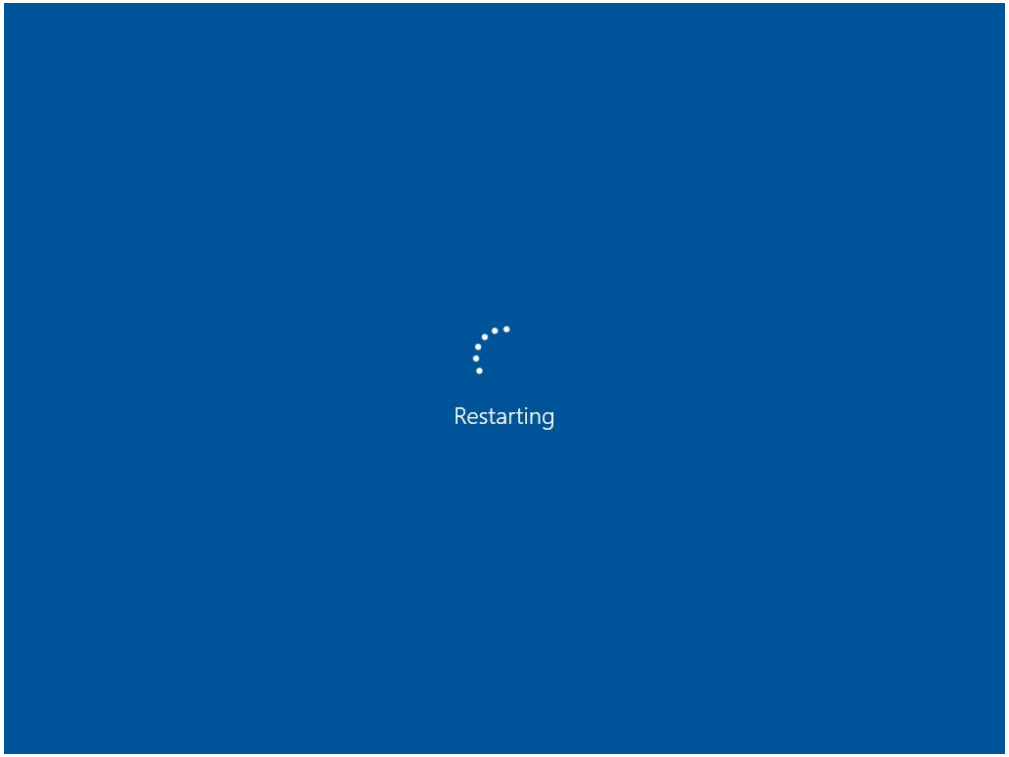
Anyone remember this?



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

In [5]:

```
class Patient:

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

```
  File "C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/2779481092.py", line 3
    def __init__(self, name)
                            ^
SyntaxError: invalid syntax
```

In [5]:

```
class Patient:

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

```
  File "C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/2779481092.py", 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

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 [7]:

```
class Patient:

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

p1 = Patient('Niels')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/4112194549.py in <module>
      5         self.ethnicity=ethnicity
      6
----> 7 p1 = Patient('Niels')

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/4112194549.py in __init__(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 [8]:
```
class Patient:

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

```
  File "C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/2779481092.py", line 3
    def __init__(self, name)
                            ^
SyntaxError: invalid syntax
```

```
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

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 [9]:
```
a = 1
a
```

Out[9]:
```
1
```

NameError

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

In [9]:
```
a = 1
a
```

Out[9]:
```
1
```

In [10]:
```
d
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/3161387801.py in <module>
----> 1 d

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

```
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

IndexError

IndexError

```
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

IndentationError

IndentationError

In [12]:

```
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 [13]:
```
phonebook = {'Anton': 12345, 'Bettina': 23456, 'Marijn': 34567}
print(phonebook['Anton'])
print(phonebook['Niels'])
```

```
12345
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/301003906.py in <module>
      1 phonebook = {'Anton': 12345, 'Bettina': 23456, 'Marijn': 34567}
      2 print(phonebook['Anton'])
----> 3 print(phonebook['Niels'])

KeyError: 'Niels'
```

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 [14]:
```
import tensorflow
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/4294963926.py in <module>
----> 1 import tensorflow

ModuleNotFoundError: No module named 'tensorflow'
```

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

NEVER HAVE I FELT SO CLOSE TO ANOTHER SOUL

AND YET SO HELPLESSLY ALONE

AS WHEN I GOOGLE AN ERROR

AND THERE'S ONE RESULT

A THREAD BY SOMEONE WITH THE SAME PROBLEM

AND NO ANSWER

LAST POSTED TO IN 2003

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 [16]:
```
a = input('Enter a value here: ')
type(a)
```

```
Enter a value here: 10
```

Out[16]:
```
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 [17]:
```
a = input('Give me an integer: ')

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

```
Give me an integer: 10
```

Out[17]:
```
int
```

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 [18]:
```
while True:
    try:
        a = 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: ', a)
```

```
Please enter a number: a
Oops!  That was no valid number.  Try again...
Please enter a number: abc
Oops!  That was no valid number.  Try again...
Please enter a number: 9.5
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

Catching multiple exceptions

Try/Except: Example 2

Catching multiple exceptions

In [19]:

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

while True:
    try:
        a = int(input('Please enter an integer position to index in four_names: '))
        indexed_name = four_names[a]
        break
    except ValueError:
        print('Oops!  That was no valid number.  Try again...')
    except IndexError:
        print('There is no name at that position in the list. Try again...')

print('You (finally) entered a valid number and it was: ', a)
print('And that position existed in the list and corresponds with name: ', indexed_name)
```

```
Please enter an integer position to index in four_names: abc
Oops!  That was no valid number.  Try again...
Please enter an integer position to index in four_names: 6
There is no name at that position in the list. Try again...
Please enter an integer position to index in four_names: 4
There is no name at that position in the list. Try again...
Please enter an integer position to index in four_names: 0
You (finally) entered a valid number and it was:  0
And that position existed in the list and corresponds with name:  Visara
```

Try/Except: Example 3

File reading and accessing the error object

Try/Except: Example 3

File reading and accessing the error object

In [20]:
```
# 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 [21]:

```
try:
    # open a file from disk, like this notebook
    with open('Lecture06-ErrorsAndExceptionsWITH-A-TYPE-IN-THE-FILENAME.ipynb', 'r') as f
        # print the first part
        print(f.read()[:50])

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-ErrorsAndExceptionsWITH-A-
TYPE-IN-THE-FILENAME.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 [22]:
```
class Patient:

    def __init__(self, name, gender, age):
        self.name=name
        self.gender=gender
        self.updateAge(age) # <-- NOTE that the updateAge method is already used here to

    # 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_11144/2954757338.py in <module>
     14
     15 p1 = Patient('Niels', 'M', '39')
---> 16 p1.age

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

One way to do this is by using if/else

In [22]:
```
class Patient:

    def __init__(self, name, gender, age):
        self.name=name
        self.gender=gender
        self.updateAge(age) # <-- NOTE that the updateAge method is already used here to

    # 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_11144/2954757338.py in <module>
     14
     15 p1 = Patient('Niels', 'M', '39')
---> 16 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 [23]:

```
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_11144/1017788043.py in <module>
     13             raise ValueError("Age must be a number! Age was not updated. ")
     14
---> 15 p1 = Patient('Niels', 'M', '39')
     16 p1.age

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/1017788043.py in __init__(self, name, gender, age)
      4         self.name=name
      5         self.gender=gender
----> 6         self.updateAge(age)
      7
      8     # method for updating the age of the patient
```

```
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/1017788043.py in u
pdateAge(self, age)
     11             self.age=age
     12         else:
---> 13             raise ValueError("Age must be a number! Age was not up
dated. ")
     14
     15 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 [24]:
```
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_11144/3912199431.py in <module>
     13         self.age=age
     14
---> 15 p1 = Patient('Niels', 'M', '39')
     16 p1.age

C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/3912199431.py in __init__(self, name, gender, age)
      4         self.name=name
      5         self.gender=gender
----> 6         self.updateAge(age)
      7
      8     # method for updating the age of the patient
```

```
C:\Users\NIELS~1.HAM\AppData\Local\Temp/ipykernel_11144/3912199431.py in u
pdateAge(self, age)
      9     def updateAge(self, age):
     10
---> 11         assert (type(age) == int), "Age is not an integer" # note:
message describes what happens when the statement fails
     12         # if the assert statement passes, the code keeps running,
otherwise an AssertionError is raised
     13         self.age=age

AssertionError: Age is not an integer
```

Takeaway messages

Don’t be scared of errors.

Read them. Google them.

Takeaway messages

Don’t be scared of errors.

Read them. Google them.

Errors’ purpose:

```
* A graceful exit
* Separate what's supposed to happen, from what can go wrong
```

Takeaway messages

Don't be scared of errors.

Read them. Google them.

Errors' purpose:

```
* A graceful exit
* Separate what's supposed to happen, from what can go wrong
```

Handle exceptions by using `try` and `except`

Takeaway messages

Don't be scared of errors.

Read them. Google them.

Errors' purpose:

```
* A graceful exit
* Separate 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