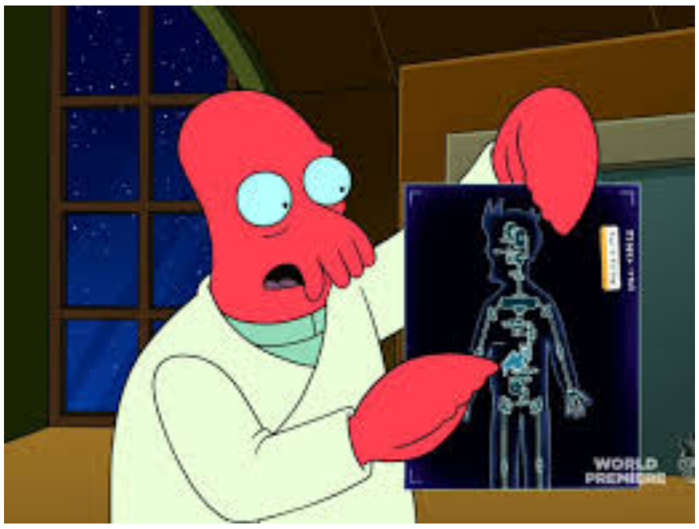
"Nice-to-have" knowledge

with regards to the test but likely important for your later professional lives.



What is a program?

```
    Input      ---->      Processing      ---->      Output
```

But there are various forms of output...

```
    Input      ---->      Processing      ---->      Output
                               |
                               +-------------->      Exception
```

In [1]:

```
3 + 'Hej, you :)'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-bb0036e67db9> in <module>
----> 1 3 + 'Hej, you :)'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Exception (nice-to-have)

- An unexpected behaviour that terminates the program
    - Unless handled

In [1]:

```python
try:
    3 + 'Hej, you :)'
except:
    print('You cannot add strings to integers, what should that be?')

print('here')
```

```
You cannot add strings to integers, what should that be?
here
```

Recipe for writing code: waterfall model

1. Figure out what you want (requirements)

2. Figure out what you **really** want (pseudo-code)

3. Write a program that fulfills the requirements

4. Test if your code fulfills the requirements

Agile development

- Agile manifesto: http://agilemanifesto.org/ (http://agilemanifesto.org/)

> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan

Development driven by tests

- You know now how hard it is to write code
- How do you make sure that your code is doing *what you think* your code is doing?

Testing

- A way to remove errors or at least make them less probable
- Tests *assert* that your code works *like you think* it works
- When you add new code, old tests make sure that nothing breaks (regression)

Your turn!

- Write a function `german_polite_form` that takes a name and prepends `'Sehr geehrte Frau '` to that name before returning it.

In [ ]:

```python
def german_polite_form(name):
    return 'Sehr geehrte Frau ' + name


print(german_polite_form('Ada Lovelace'))
```

- Run the function with the argument `'Ada Lovelace'`
- Run the function with the argument `'Hansi Hinterseer'`
- Run the function with the argument `3`

What happens for each of the input values?

Testing flow chart

```
What is the output?     ---->     Exactly what I wanted     ---->     Goo
d!
```

```
      |
      +---------------------->     Unexpected output         ---->   Fix i
t!
```

```
      |
      +---------------------->     Code exception (assumption breaking)
```

```
                                          |
                                          +---->   Assumption is sound     ---
    ->     Blame user
```

```
                                          |
                                          +---->   Assumption is bad       ---
    ->     Fix it!
```

Unit Tests

Unit tests are small programs that test for correctness of specific aspects of the smallest units of your program, which are either functions or methods.

In [2]:

```python
import random
import us_names


def generate_names(gender, number):
    """Generates a list of names, which are randomly created out
    of names from the US census 1990.

    :param gender: str
        The gender of the name. Can be 'female' or 'male'
    :param number: int
        Amount of names in the returned list

    :return: list
        A list of strings with either female or male US names.
    """
    all_names = []
    if gender == 'female':
        names = us_names.FEMALE_NAMES
    elif gender == 'male':
        names = us_names.MALE_NAMES
    else:
        print("Error: Gender should be either 'female' or 'male'")
    for _ in range(number):
        name = random.choice(names)
        surname = random.choice(us_names.SURNAMES)
        fullname = name + ' ' + surname
        all_names.append(fullname)
    return all_names
```

How to test this program?

Probably something like this:

In [3]:

```python
print(generate_names('female', 10))
print(generate_names('male', 5))
print(generate_names('female', 20))
print(generate_names('male', 25))
```

```
['Alia Yarmitsky', 'Tristan Utz', 'Denae Harrower', 'Sherryl Douin',
'Polly Adolphe', 'Joana Cullar', 'Dreama Videtto', 'Dorthey Reddy',
'Earlie Quitedo', 'Lecia Sybounheuan']
['Lewis Spruce', 'Rocco Wiers', 'Brian Lesmeister', 'Jamie Sconce',
'Kent Tyler']
['Bettie Tohill', 'Zina Korbel', 'Eladia Thigpen', 'Pilar Merriam',
'Lakendra Wakayama', 'Beth Amodei', 'Johanne Cilenti', 'Kimiko Cotty
', 'Sharonda Massingale', 'Danille Slocum', 'Demetra Trover', 'Dusti
Landmesser', 'Clora Ballagh', 'Rolanda Capellas', 'Anastacia Stockbr
idge', 'Vinita Vallone', 'Nella Bremme', 'Nanette Kowing', 'Tobi Har
bin', 'Johnna Hawelu']
['Tyree Willard', 'Dorian Lavani', 'Mckinley Parsley', 'Tyrone Goodp
astor', 'Rex Bellish', 'Wm Lieblong', 'Jonathon Rinaldi', 'Jude Soff
er', 'Weston Spinoso', 'Freddie Kissane', 'Alonso Dunnahoo', 'Quincy
Garzone', 'Abraham Eckmann', 'James Cocke', 'Ward Wykes', 'Ike Penig
ar', 'Chi Nitcher', 'Nathan Sena', 'Thad Kidner', 'Noe Hrovat', 'Dam
ion Cason', 'Alex Luyando', 'Andre Kishimoto', 'Dalton Deranick', 'C
lifton Renteria']
```

But did you think about weird input that some other programmer might use?

In [7]:

```python
generate_names('schnippschnapp', 8)
generate_names(-3, 8)
generate_names('male', 123456789123456789123456789123456789123456789123456789
```

```
Error: Gender should be either 'female' or 'male'

---------------------------------------------------------------------------
--------
UnboundLocalError                         Traceback (most recent cal
l last)
<ipython-input-7-da6d9d62b72b> in <module>
----> 1 generate_names('schnippschnapp', 8)
      2 generate_names(-3, 8)
      3 generate_names('male', 12345678912345678912345678912345678912
23456789123456789123456789123456789123456789)

<ipython-input-1-42a114a18956> in generate_names(gender, number)
     23         print("Error: Gender should be either 'female' or 'm
ale'")
     24     for _ in range(number):
---> 25         name = random.choice(names)
     26         surname = random.choice(us_names.SURNAMES)
     27         fullname = name + ' ' + surname

UnboundLocalError: local variable 'names' referenced before assignme
nt
```

This is what test cases with many unit tests are for.

You just specify in another file, which you call `test_<program_to_test_name>.py` and in it you specifiy your unit tests.

In [16]:

```python
from generate_names import generate_names


def test_generate_names():
    names = generate_names('schnippschnapp', 8)
    assert len(names) == 0

```

In [4]:

```python
assert True
```

In [5]:

```
assert False
```

```
---------------------------------------------------------------------
--------
AssertionError                            Traceback (most recent cal
l last)
<ipython-input-5-a871fdc9ebee> in <module>
----> 1 assert False

AssertionError:
```

`assert` ?

In essence the `assert expression` statement does the following:

```
if not expression:
    raise AssertionError
```

https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement (https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement)

Executing each unit test manually is tedious. Consequenlty, we use a testing framework `py.test` , which automates the process of running a set of unit tests.

You can run your tests from the command-line by pointing `pytest` to the file containing your unit tests.

```
$ pytest test_generate_names.py
```

It will collect all functions that start with a `test_` , execute them sequentially, and report if the unit test fails or passes.

```
$ pytest test_generate_names.py
======================================== test session starts ============
============================
platform darwin -- Python 3.7.1, pytest-4.0.2, py-1.7.0, pluggy-0.8.0
rootdir: /Users/ropf/Documents/Lectures/qualification-seminar-2019/sessio
n-7, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.1, doctestplus-0.2.0, arraydiff-
0.3
collected 4 items

test_generate_names.py ..FF
[100%]

============================================= FAILURES ==================
============================
...
```

Test Case

A *test case* is a collection of unit tests that together prove that a function behaves as it's supposed to, within the full range of situations you expect it to handle.

A good test case considers all the possible kinds of input a function could receive and includes tests to represent each of these situations.

Test-driven Development

In Test-driven Development (TDD) you start by writing your test before writing your actual program.

The idea is, that you -or one of your friends/colleagues- specifies the input a function/method requires and the output it is supposed to create.

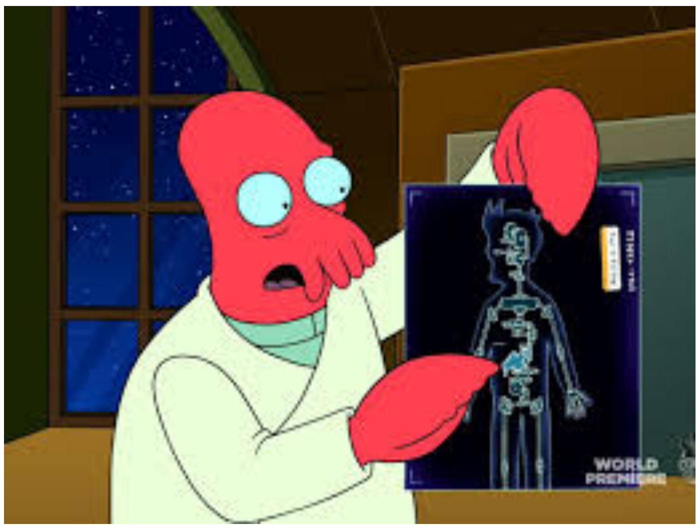
Then you implement the functionality until all given unit tests pass. That should mean that your code does at least what it is asserted to do.

Unit Testing Exercise

- Take your previous function `german_polite_form` and save it in the file `german.py`
- Create the file `test_german.py`
- Write one test that verifies that `Conchita Wurst` gets addressed correctly
  - You only need to import your `german_polite_form` file and create a function for the test starting with `test_`
  - Use `assert` to test your assumption
- Write one test that verifies that using the number 3 does *not* work
  - Use the `try ... except` construct

"Nice-to-have" knowledge

with regards to the test but likely important for your later professional lives.



What is a program?

```
Input      ---->      Processing      ---->      Output
```

But there are various forms of output...

```
Input      ---->      Processing      ---->      Output
                           |
                           +-------------->      Exception
```

```
In [1]:
```

```
---------------------------------------------------------------------
--------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-1-bb0036e67db9> in <module>
----> 1 3 + 'Hej, you :)'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Exception (nice-to-have)

- An unexpected behaviour that terminates the program
    - Unless handled

```
In [1]:
```

```
You cannot add strings to integers, what should that be?
here
```

Recipe for writing code: waterfall model

1. Figure out what you want (requirements)

2. Figure out what you **really** want (pseudo-code)

3. Write a program that fulfills the requirements

4. Test if your code fulfills the requirements

Agile development

- Agile manifesto: [http://agilemanifesto.org/ (http://agilemanifesto.org/)](http://agilemanifesto.org/)

> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan

Development driven by tests

- You know now how hard it is to write code
- How do you make sure that your code is doing *what you think* your code is doing?

Testing

- A way to remove errors or at least make them less probable
- Tests *assert* that your code works *like you think* it works
- When you add new code, old tests make sure that nothing breaks (regression)

Your turn!

- Write a function `german_polite_form` that takes a name and prepends `'Sehr geehrte Frau '` to that name before returning it.

```
In [ ]:
```

- Run the function with the argument `'Ada Lovelace'`
- Run the function with the argument `'Hansi Hinterseer'`
- Run the function with the argument `3`

What happens for each of the input values?

Testing flow chart

```
What is the output?     ---->     Exactly what I wanted     ---->     Goo
d!
```

```
      |
      +---------------------->     Unexpected output         ---->   Fix i
t!
```

```
      |
      +---------------------->     Code exception (assumption breaking)
```

```
                                        |
                                        +----->   Assumption is sound     ---
->     Blame user
```

```
                                        |
                                        +----->   Assumption is bad       ---
->     Fix it!
```

Unit Tests

Unit tests are small programs that test for correctness of specific aspects of the smallest units of your program, which are either functions or methods.

In [2]:

```

```

How to test this program?

Probably something like this:

In [3]:

```
['Alia Yarmitsky', 'Tristan Utz', 'Denae Harrower', 'Sherryl Douin',
'Polly Adolphe', 'Joana Cullar', 'Dreama Videtto', 'Dorthey Reddy',
'Earlie Quitedo', 'Lecia Sybounheuan']
['Lewis Spruce', 'Rocco Wiers', 'Brian Lesmeister', 'Jamie Sconce',
'Kent Tyler']
['Bettie Tohill', 'Zina Korbel', 'Eladia Thigpen', 'Pilar Merriam',
'Lakendra Wakayama', 'Beth Amodei', 'Johanne Cilenti', 'Kimiko Cotty
', 'Sharonda Massingale', 'Danille Slocum', 'Demetra Trover', 'Dusti
Landmesser', 'Clora Ballagh', 'Rolanda Capellas', 'Anastacia Stockbr
idge', 'Vinita Vallone', 'Nella Bremme', 'Nanette Kowing', 'Tobi Har
bin', 'Johnna Hawelu']
['Tyree Willard', 'Dorian Lavani', 'Mckinley Parsley', 'Tyrone Goodp
astor', 'Rex Bellish', 'Wm Lieblong', 'Jonathon Rinaldi', 'Jude Soff
er', 'Weston Spinoso', 'Freddie Kissane', 'Alonso Dunnahoo', 'Quincy
Garzone', 'Abraham Eckmann', 'James Cocke', 'Ward Wykes', 'Ike Penig
ar', 'Chi Nitcher', 'Nathan Sena', 'Thad Kidner', 'Noe Hrovat', 'Dam
ion Cason', 'Alex Luyando', 'Andre Kishimoto', 'Dalton Deranick', 'C
lifton Renteria']
```

But did you think about weird input that some other programmer might use?

In [7]:

```
Error: Gender should be either 'female' or 'male'

---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent cal
l last)
<ipython-input-7-da6d9d62b72b> in <module>
----> 1 generate_names('schnippschnapp', 8)
      2 generate_names(-3, 8)
      3 generate_names('male', 12345678912345678912345678912345678912
3456789123456789123456789123456789123456789)

<ipython-input-1-42a114a18956> in generate_names(gender, number)
     23         print("Error: Gender should be either 'female' or 'm
ale'")
     24     for _ in range(number):
---> 25         name = random.choice(names)
     26         surname = random.choice(us_names.SURNAMES)
     27         fullname = name + ' ' + surname
```

This is what test cases with many unit tests are for.

You just specify in another file, which you call `test_<program_to_test_name>.py` and in it you specifiy your unit tests.

In [16]:

```
```

In [4]:

```
```

In [5]:

```
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-5-a871fdc9ebee> in <module>
----> 1 assert False

AssertionError: 
```

`assert` ?

In essence the `assert expression` statement does the following:

```
if not expression:
    raise AssertionError
```

https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement
(https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement)

Executing each unit test manually is tedious. Consequenlty, we use a testing framework `py.test` , which automates the process of running a set of unit tests.

You can run your tests from the command-line by pointing `pytest` to the file containing your unit tests.

```
$ pytest test_generate_names.py
```

It will collect all functions that start with a `test_` , execute them sequentially, and report if the unit test fails or passes.

```
$ pytest test_generate_names.py
======================================== test session starts ============
============================
platform darwin -- Python 3.7.1, pytest-4.0.2, py-1.7.0, pluggy-0.8.0
rootdir: /Users/ropf/Documents/Lectures/qualification-seminar-2019/sessio
n-7, inifile:
plugins: remotedata-0.3.1, openfiles-0.3.1, doctestplus-0.2.0, arraydiff-
0.3
collected 4 items

test_generate_names.py ..FF
[100%]

============================================= FAILURES ==================
============================
...
```

Test Case

A *test case* is a collection of unit tests that together prove that a function behaves as it's supposed to, within the full range of situations you expect it to handle.

A good test case considers all the possible kinds of input a function could receive and includes tests to represent each of these situations.

Test-driven Development

In Test-driven Development (TDD) you start by writing your test before writing your actual program.

The idea is, that you -or one of your friends/colleagues- specifies the input a function/method requires and the output it is supposed to create.

Then you implement the functionality until all given unit tests pass. That should mean that your code does at least what it is asserted to do.

Unit Testing Exercise

- Take your previous function `german_polite_form` and save it in the file `german.py`
- Create the file `test_german.py`
- Write one test that verifies that `Conchita Wurst` gets addressed correctly
  - You only need to import your `german_polite_form` file and create a function for the test starting with `test_`
  - Use `assert` to test your assumption
- Write one test that verifies that using the number 3 does *not* work
  - Use the `try ... except` construct