Debugging

Bad things that can happen to your code

- Syntax Errors: Prevent your code from running (i.e. pre-runtime)
- Runtime Error: Occur during runtime (Exception)
- Semantic Error: Code runs, but not the way you like (Bugs)

Bad things that can happen to your code

- Syntax Errors: Prevent your code from running (i.e. pre-runtime)
- Runtime Error: Occur during runtime (Exception)
- Semantic Error: Code runs, but not the way you like (Bugs)
- Which one of these is a syntax error, which one is a bug, and which one will throw an exception?
  1. Attempting to divide by 0
  2. Not closing a parenthesis
  3. Not dividing by 100 when computing a percentage

Don't be scared

```
Traceback (most recent call last):
  File "./test.py", line 21, in <module>
    main()
  File "./test.py", line 14, in main
    data=tips, legend=False)
  File "/usr/local/lib/python3.6/dist-packages/seaborn/rela
    **plot_kws)
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/_
    return func(ax, *args, **kwargs)
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/a
    collection.update(kwargs)
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/a
    ret = [_update_property(self, k, v) for k, v in props.i
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/a
    ret = [_update_property(self, k, v) for k, v in props.i
  File "/usr/local/lib/python3.6/dist-packages/matplotlib/a
    raise AttributeError('Unknown property %s' % k)
AttributeError: Unknown property xcol
```

Error messages are your friend

- You will spend most of the time debugging
- It's detective work: Where does the bug come from, how to fix it w/o breaking other things
- Tracebacks help you: What kind of error, where (approximately)
  - Example from above: `AttributeError in line 14`

Most important rules while debugging

- Write easy code
- Use print statements to verify objects
- Experiment to check your hypotheses
- Do scaffolding: Write, check, repeat (Get something working and keep it working)
- Think formally (unlike in natural languages)
  - No ambiguity
  - Less redundancy
  - Always literal

Most important rules while debugging

- Write easy code
- Use print statements to verify objects
- Experiment to check your hypotheses
- Do scaffolding: Write, check, repeat (Get something working and keep it working)
- Think formally (unlike in natural languages)
  - No ambiguity
  - Less redundancy
  - Always literal
- The problem always sits behind the keyboard

Inspecting the object

```
1 my_list = {'syntax': 10, 'runtime': 99}
2 print(type(my_list))
```

Inspecting the object

```
1 my_list = {'syntax': 10, 'runtime': 99}
2 print(type(my_list))
```

- What is the type of object `my_list`?

Checking the version

Every decent package has a magic attribute `.__version__`:

```
from import pandas as pd

pd.__version__
```

Useful to check whether your version might contain bugs

Know your error I

```
1  x = "90"
2  y = 100
3  z = x + y
```

Know your error I

```
1  x = "90"
2  y = 100
3  z = x + y
```

- TypeError: you try to combine two objects that are not compatible

Know your error II

```
1 currencies = ["dollar", "euro"]
2 print(currency)
```

Know your error II

```
1 currencies = ["dollar", "euro"]
2 print(currency)
```

- NameError: you refer to an object that does not exist

Know your error III

```
1  int("9.0")
```

Know your error III

```
1 int("9.0")
```

- ValueError: the value you passed to a parameter does not pass the function's limitations on the value

Know your error IV

```
1 marks = [1, 1, 4, 3, 6]
2 print(marks[5])
```

Know your error IV

```
1 marks = [1, 1, 4, 3, 6]
2 print(marks[5])
```

- IndexError: you are referring to an element in a container that does not exist

Know your error V

```
1 capitals = {'germany': 'berlin', 'austria': 'vienna'}
2 print(capitals['france'])
```

Know your error V

```
1 capitals = {'germany': 'berlin', 'austria': 'vienna'}
2 print(capitals['france'])
```

- KeyError: you are referring to a key in a dict (or dict-like object) that does not exist

Know your error VI

```
1 my_list = "fbcead"
2 my_list.sort()
```

Know your error VI

```
1 my_list = "fbcead"
2 my_list.sort()
```

- AttributeError: what you want to do with an object is not possible (mostly: the object is not what you think it is)

Handling exceptions

- Sometimes there might be anticipated changes to your object causing Exceptions
- It is generally cheaper to use a try-except block than to check whether subsequent code will work
- ... unless your error occurs more than 50% of the time
- General rule: Catch only specific errors!

```
1     average = sum(a_list) / len(a_list)
```

What when `a_list` is empty 10% of the time?

Warnings

- Warnings are messages only
- Warnings do not break runtime
- Most of the time you have DeprecationWarnings and pandas' https://www.dataquest.io/blog/settingwithcopywarning/SettingwithCopyWarning