SNAKES!

Python is an interpreted, dynamically-typed, high-level, garbage-collected, object-oriented-functional-imperative, and widely used scripting language.

- **Interpreted:** instructions executed without being compiled into (virtual) machine instructions*
- **Dynamically-typed:** verifies type safety at runtime
- **High-level:** abstracted away from the raw metal and kernel
- **Garbage-collected:** memory management is automated
- **OOFI:** you can do bits of OO, F, and I programming

Not the point of this class!

- Python is fast (developer time), intuitive, and used in industry!

*you can compile Python source, but it's not required

THE ZEN OF PYTHON

- 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.

Thanks: SDSMT ACM/LUG

LITERATE PROGRAMMING

Literate code contains in one document:

- the source code;
- text explanation of the code; and
- the end result of running the code.

Basic idea: present code in the order that logic and flow of human thoughts demand, not the machine-needed ordering

- Necessary for data science!
- Many choices made need textual explanation, ditto results.

IP[y]: IPython Interactive Computing

jupyter

10-MINUTE PYTHON PRIMER

Define a function:

```
def my_func(x, y):
    if x > y:
        return x
    else:
        return y
```

Define a function that returns a tuple:

```
def my_func(x, y):
    return (x-1, y+2)

(a, b) = my_func(1, 2)
```

```
a = 0; b = 4
```

USEFUL BUILT-IN FUNCTIONS: COUNTING AND ITERATING

`len`: returns the number of items of an enumerable object

```
len( [‘c’, ‘m’, ‘s’, ‘c’, 3, 2, 0] )
```

```
7
```

`range`: returns an iterable object

```
list( range(10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`enumerate`: returns iterable tuple (index, element) of a list

```
enumerate( [“311”, “320”, “330”] )
```

```
[(0, “311”), (1, “320”), (2, “330”)]
```

https://docs.python.org/3/library/functions.html

USEFUL BUILT-IN FUNCTIONS: MAP AND FILTER

`map`: apply a function to a sequence or iterable

```
arr = [1, 2, 3, 4, 5]
map(lambda x: x**2, arr)
```

```
[1, 4, 9, 16, 25]
```

`filter`: returns a list of elements for which a predicate is true

```
arr = [1, 2, 3, 4, 5, 6, 7]
filter(lambda x: x % 2 == 0, arr)
```

```
[2, 4, 6]
```

We’ll go over in much greater depth with pandas/numpy.

PYTHONIC PROGRAMMING

Basic iteration over an array in Java:

```
int[] arr = new int[10];
for(int idx=0; idx<arr.length; ++idx) {
    System.out.println( arr[idx] );
}
```

Direct translation into Python:

```
idx = 0
while idx < len(arr):
    print( arr[idx] ); idx += 1
```

A more “Pythonic” way of iterating:

```
for element in arr:
    print( element )
```

LIST COMPREHENSIONS

Construct sets like a mathematician!

- $P = \{ 1, 2, 4, 8, 16, \ldots, 2^{16} \}$
- $E = \{ x \mid x \text{ in } \mathbb{N} \text{ and } x \text{ is odd and } x < 1000 \}$

Construct lists like a mathematician who codes!

```
P = [ 2**x for x in range(17) ]
```

```
E = [ x for x in range(1000) if x % 2 != 0 ]
```

Very similar to `map`, but:

- **You’ll see these way more than `map` in the wild**
- **Many people consider `map`/`filter` not “pythonic”**
- **They can perform differently (`map` is “lazier”)**

EXCEPTIONS

Syntactically correct statement throws an exception:

- `tweepy` (Python Twitter API) returns “Rate limit exceeded”
- sqlite (a file-based database) returns `IntegrityError`

```
print('Python', python_version())

try:
   cause_a_NameError
except NameError as err:
   print(err, '-> some extra text')
```

PYTHON 2 VS 3

Python 3 is intentionally backwards incompatible

- (But not *that* incompatible)

Biggest changes that matter for us:

- `print “statement”` → `print(“function”)`
- `1/2 = 0` → `1/2 = 0.5` and `1//2 = 0`
- ASCII `str` default → default Unicode

Namespace ambiguity fixed:

```
i = 1
[i for i in range(5)]
print(i)   # ????????
```

TO ANY CURMUDGEONS …

If you’re going to use Python 2 anyway, use the `_future_` module:

- Python 3 introduces features that will throw runtime errors in Python 2 (e.g., `with` statements)
- `_future_` module incrementally brings 3 functionality into 2
- https://docs.python.org/2/library/__future__.html

```
from _future_ import division
from _future_ import print_function
from _future_ import please_just_use_python_3
```

EXTRA RESOURCES

Plenty of tutorials on the web:

- https://www.learnpython.org/