Python for Data and Text Mining

•••

Mohammed Shameer Iqbal

bit.ly/SMU-2019-1

Who am I?

- Senior AI developer at Rein Tech - rein.ai
- Founder of InsightWell
- Alumni of Acadia University
- Started writing Python six years ago as I hated Matlab®, never turned back
- I write Python on most days but do write JavaScript when I hate myself
- Make a lot of bad jokes

Agenda

- Introduction to python: history, philosophy
- Hello World!
- Whitespaces, no brackets
- Data types
- Operators
- Data structures
- Standard libraries
- Numpy
- Leave some time for question and clarification

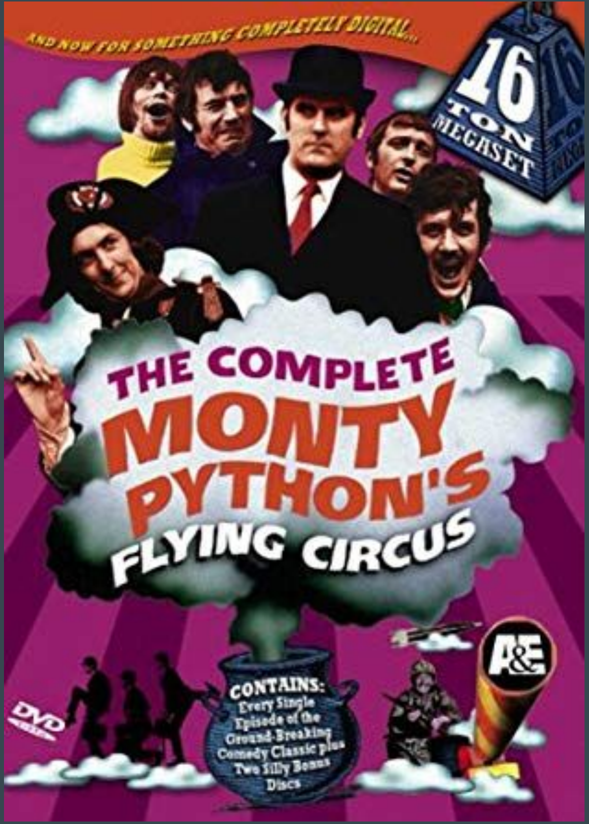
Workshop != Lecture

- Follow along, ask me to slow down if I go too fast
- Ask questions
- Ask for help, flag either me or one of the TAs down
- Basis for upcoming workshop and classes

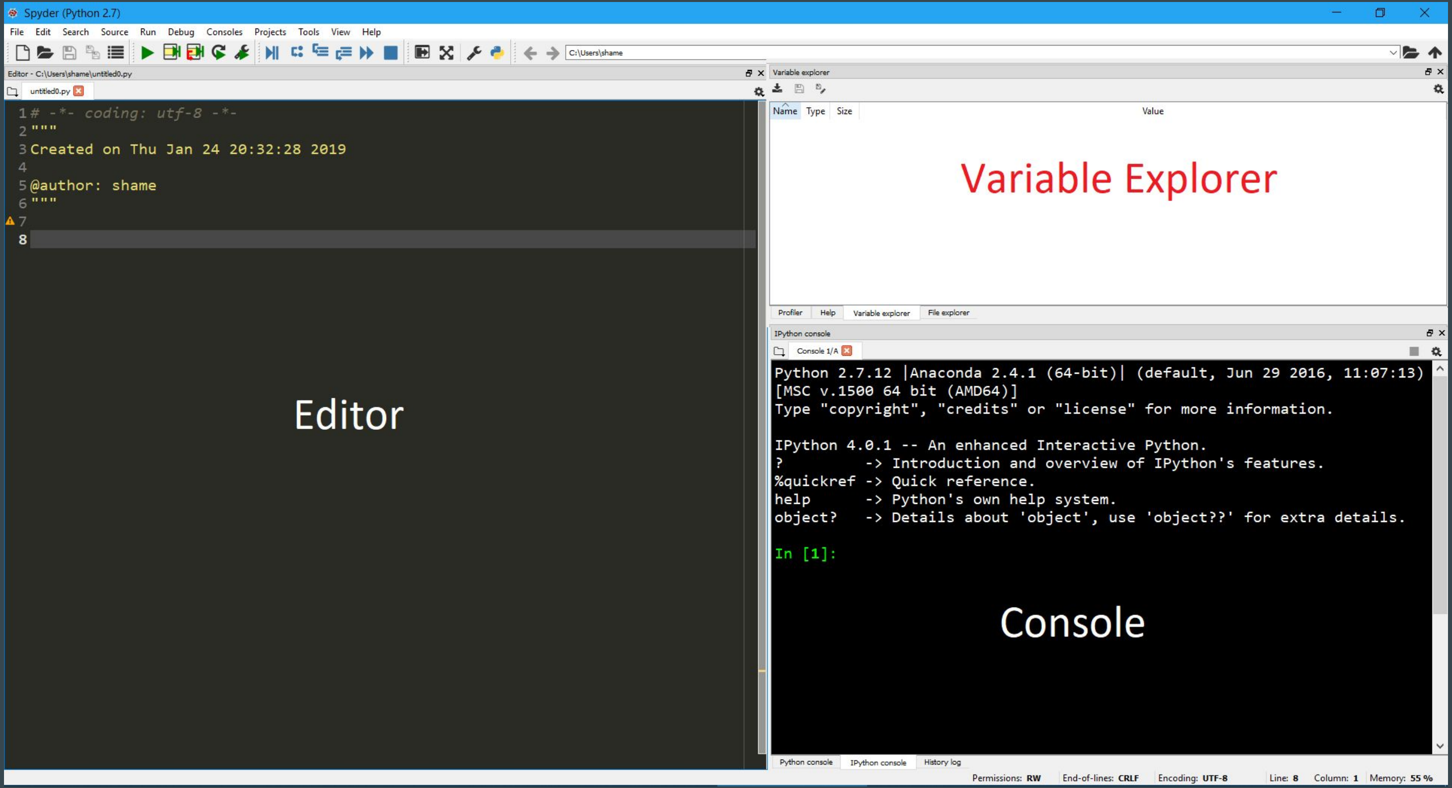
SPYDER

Python History

- Created by Guido van Rossum in 1991
- Older than Java
- Named after British TV comedy Monty Python
- Current python version is 3.7
- Support for python 2 will end in 2020


AND NOW FOR SOMETHING COMPLETELY DIGITAL...
16 TON MEGASET
THE COMPLETE MONTY PYTHON'S FLYING CIRCUS
CONTAINS: Every Single Episode of the Ground-Breaking Comedy Classic plus Two Silly Bonus Discs
A&E
DVD

Spyder Editor


Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\shame
Editor - C:\Users\shame\untitled0.py
untitled0.py
Variable explorer
Name Type Size Value
Variable Explorer
Profiler Help Variable explorer File explorer
IPython console
Console 1/A
Editor
Console
Python console IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 8 Column: 1 Memory: 55 %

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 24 20:32:28 2019

@author: shame
"""

```

```
Python 2.7.12 |Anaconda 2.4.1 (64-bit)| (default, Jun 29 2016, 11:07:13)
[MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 4.0.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Hello World

```
print("Hello, World!")
```

Python’s philosophy

- Zen of python

```
> import this
```

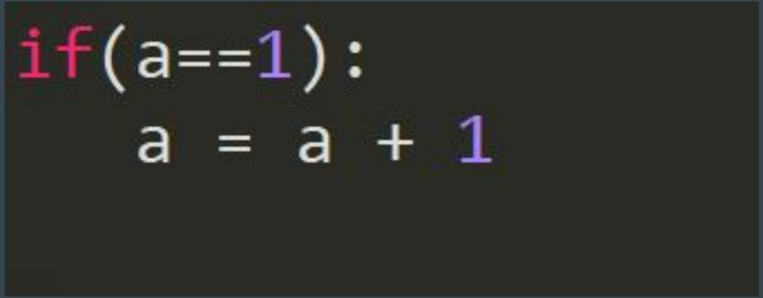
- Highlights:
    - Beautiful is better than ugly
    - Explicit is better than implicit
    - Simple is better than complex
    - Complex is better than complicated
    - Readability counts

What’s different in Python

- Interpreted
- Interactive
- Object oriented
- Dynamic - more on that later
- Simple and reads like pseudocode
- Comes with standard library for most tasks
- Common scripting language
- Automatic garbage collection (no more malloc, free or seg faults)

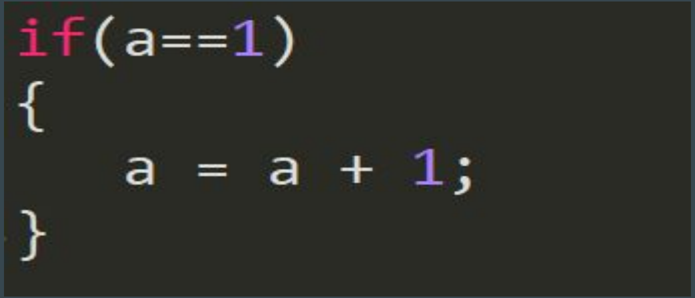
Whitespaces

- Spaces and tabs count as whitespace
- Indentation denotes a code block, no braces, no semicolons



```
if(a==1):
    a = a + 1
```

vs



```
if(a==1)
{
    a = a + 1;
}
```

- Try:

```
from __future__ import braces

File "<ipython-input-16-2aebb3fc8ecf>", line 1
    from __future__ import braces
SyntaxError: not a chance
```

Comments

- Single line comment start with ‘#’

```
# This is a comment
```

- Multi line comment start and ends with “””

```
"""
This is long
and multi line
comment
"""
```

Identifiers

- Name given to things like class, functions, variables
- Combinations of letters (a-zA-Z) , digits (0-9) and underscore (_), however cannot start with a digit
- Cannot be a keyword or use special characters
  - Valid identifiers: help, help12, Help_12, _help
  - Invalid: 1help, help-1, help#1
- Snake case is recommended:
  - e.g . this_is_a_var = 10

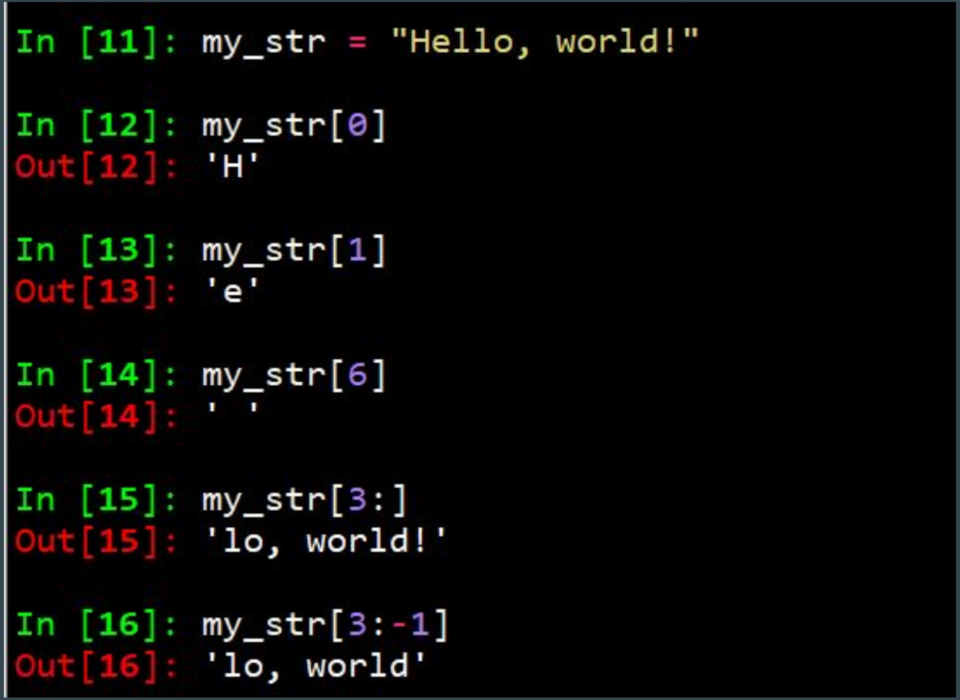
Data types

- Integers: `my_var = 1024`
- Float: `my_float_var = 1024.0`
- Boolean: `is_binary = True`
- Strings: `my_str = "Hello, world!"`
- Complex numbers, literals: let’s skip that!

Let’s talk about indices

- In a sequence such as a string, each element is a assigned an index based on their position
- Indices in Python start with “0”, this is not Matlab®
- ‘:’ is called slicing operator

“There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors”



```
In [11]: my_str = "Hello, world!"

In [12]: my_str[0]
Out[12]: 'H'

In [13]: my_str[1]
Out[13]: 'e'

In [14]: my_str[6]
Out[14]: ' '

In [15]: my_str[3:]
Out[15]: 'lo, world!'

In [16]: my_str[3:-1]
Out[16]: 'lo, world'
```

Operators

- Assignment: `=`, `+=`, `-=`
- Arithmetic: `+`, `-`, `*`, `/`, `//`, `%`, `**`
- Relational: `<`, `>`, `<=`, `>=`, `==`, `!=`
- Logical: `and`, `or`, `not`
- Membership: `in`, `not in`

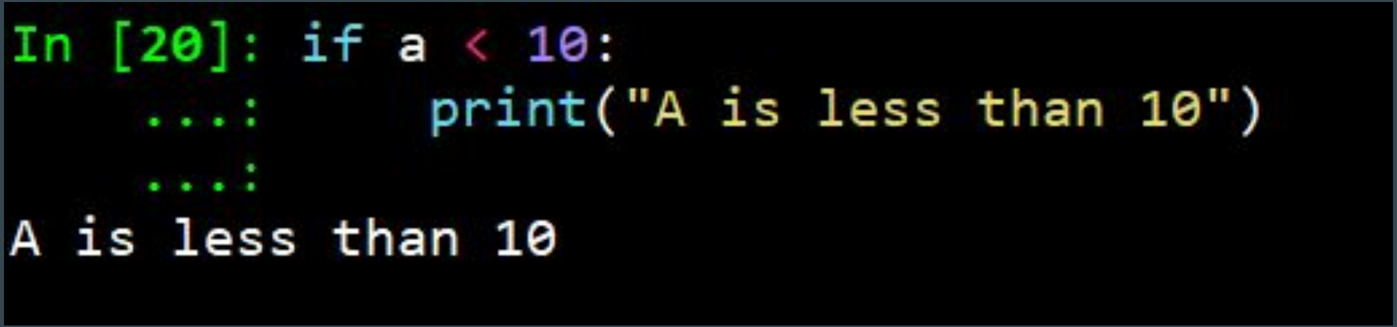
Control structures

```
if

if, elif, else

while

for - range()

break, continue, pass
```

IF

- Executes code block when condition is true:

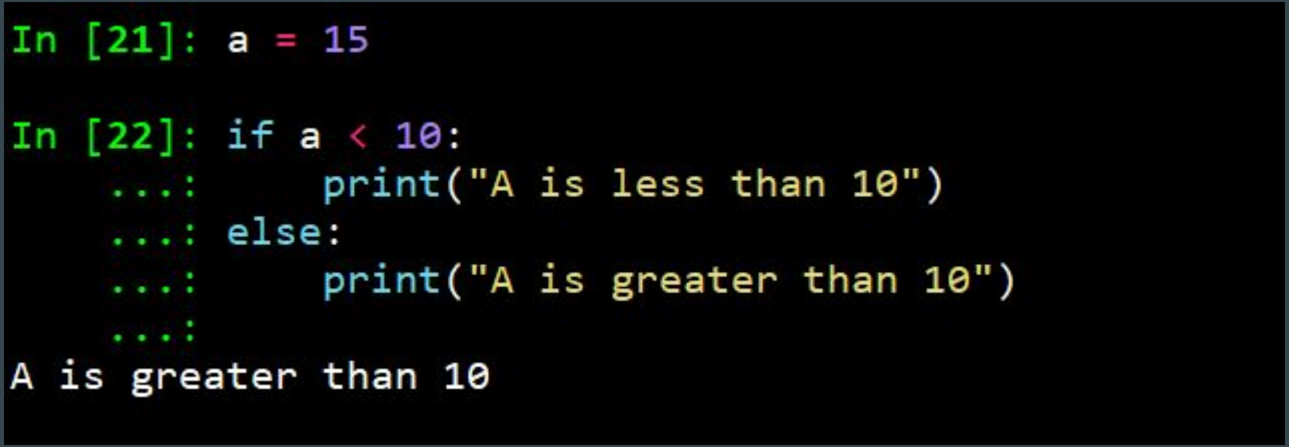
    -
      ```
      if condition:
          # to do
      ```



```
In [20]: if a < 10:
    ...:     print("A is less than 10")
    ...:
A is less than 10
```

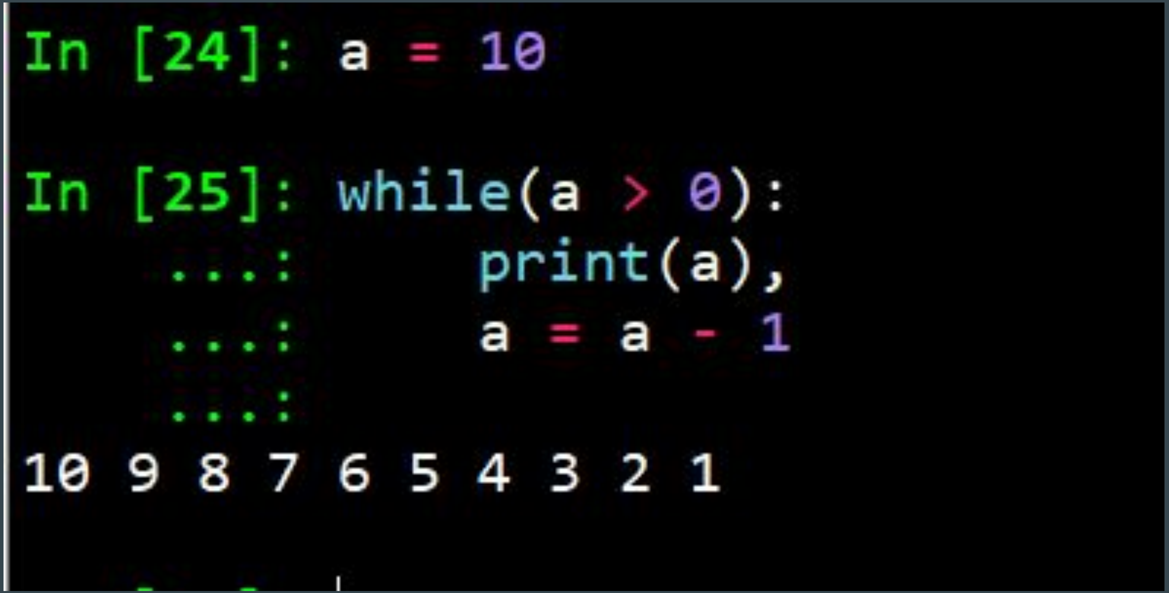
IF… ELSE

- Executes appropriate code block based on the condition:



```
In [21]: a = 15

In [22]: if a < 10:
    ...:     print("A is less than 10")
    ...: else:
    ...:     print("A is greater than 10")
    ...:
A is greater than 10
```

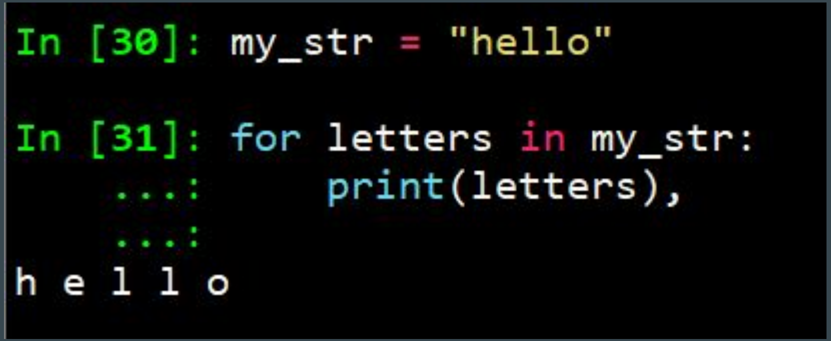
WHILE



```
In [24]: a = 10

In [25]: while(a > 0):
    ...:     print(a),
    ...:     a = a - 1
    ...:
10 9 8 7 6 5 4 3 2 1
```

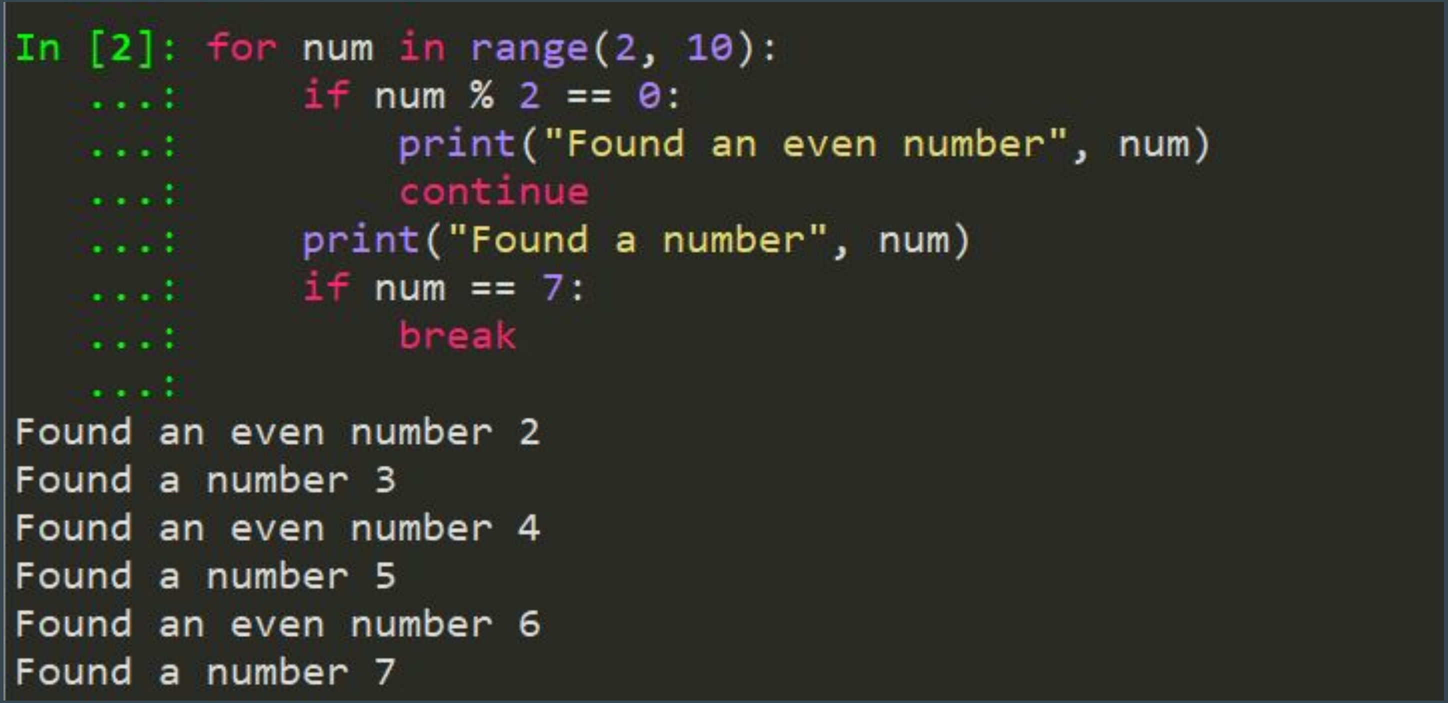
FOR

- Little different compared to for in C or other languages
- *“The Python for statement iterates over the members of a sequence in order, executing the block each time”*
- `range()` – is usually used to provide a sequence to operate for loop



```
In [30]: my_str = "hello"

In [31]: for letters in my_str:
    ...:     print(letters),
    ...:
h e l l o
```

CONTINUE... BREAK...



```
In [2]: for num in range(2, 10):
   ...:     if num % 2 == 0:
   ...:         print("Found an even number", num)
   ...:         continue
   ...:     print("Found a number", num)
   ...:     if num == 7:
   ...:         break
   ...:
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
```

Functions

- Functions are building blocks helps with code reuse, abstraction and breaks into smaller logical blocks

```
def function_name(arg):
"""
Description of what function does
arg: data type of
"""
   function code
   return something
```

Data structure

- Tuple
- Lists (stack and queues)
- Dictionary
- Sets

Tuple

- A tuple consists of a number of values separated by commas
    - `t = 12345, 54321, 'hello!'`
- Each value can be accessed using indexes
    - `t[2]`
- Immutable: cannot change value for individual elements, like strings
    - `t[2]="world"`

```
Traceback (most recent call last):
 File "<ipython-input-9-a827aea9ff96>", line 1, in <module>
   t[2]="world"
TypeError: 'tuple' object does not support item assignment
```

Lists

- List is a fundamental data structure in python
- List is a mutable data structure that can contain elements of all data types
  - `my_list = [1, "Hello", 3.4]`
- Elements can be accessed by using indices
- Unlike C, you do not have decide the list size. Just keep adding things

List functions

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

List operations

- Change the list item:
  - `my_list[1] = "world"`
- Loop through the list:
  -
    ```
    for item in my_list:
        print(item)
    ```
- Enumerate through the list:
  -
    ```
    for i, v in enumerate(my_list):
        print(i, v)
    ```

List comprehension

- a concise way to create lists

```
my_list = [i for i in range(10)]
```

  is equivalent to:

```
my_list = []
for i in range(10):
    my_list.append(i)
```

- We can include conditions as well:
  - `my_list = [i for i in "hello world" if i!='o']`

Dictionary

- *"A dictionary constant consists of a series of key-value pairs enclosed by curly braces { }"*
- Things in the dictionary are indexed based on keys
- Keys should be number or string while value could hold any type
    - `x = {"one":1, "two":2, "three":3}`
    - `my_dict = {"list_item": [1,2,3], "not_list": 4}`

Dictionary operations

- Unlike lists, dictionaries do not have positional indices
- Items has to be access through keys
  - `print(x["one"])`
- To avoid `KeyError`, use `get()` function
  - `my_dict.get("eleven", "none")`
- To add an element to dictionary, just assign value with a new key:
  - `x["four"] = 4`
- “`in`” keyword can be used to check if key exists
- `pop()` and `del()` can be used to remove an item from the dictionary

Dictionary functions

- `keys()` will return all the keys in the dictionary:

    -
      ```
      for item in x.keys():
          print(item)
      ```

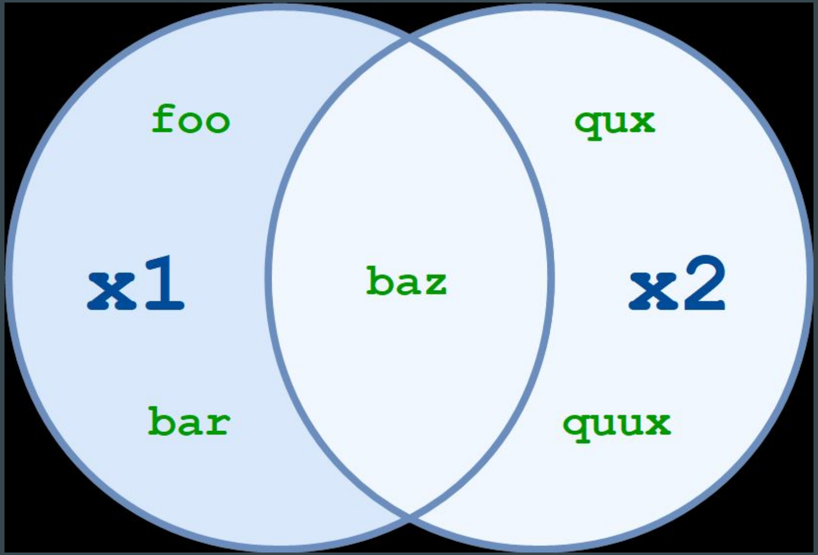
- `values()` will return the values:

    -
      ```
      for item in x.values():
          print(item)
      ```

Sets

- *"A set is an unordered collection with no duplicate elements"*
- Similar to mathematical set, we can perform set operations such as intersection, union, difference, etc.

  -
    ```
    x1 = {'foo', 'bar', 'baz'}
    x2 = {'baz', 'qux', 'quux'}
    x1.difference(x2)
    x1 - x2
    ```



Python modules

- To simplify, module in python is just a file that can contain classes, functions and variables
- Large projects are usually broken down into modules which can be reused just like we do with functions
  - `import module1`
  - `from module1 import func`
- Be careful with name spaces
- Refactor your code to make it reusable as modules
  - `if __name__ == '__main__':`

Notable standard libraries and functions

- len() - returns the length of the sequence
- re - built-in regex module
- datetime, time - date and time related functions
- os - os related functions. Very helpful in writing scripts as the functions are os agnostic
- argparse — Parser for command-line options
- random - library for pseudo-random functions
- csv - library to read and write csv files

Reference: https://docs.python.org/3/library/

Random library

-
  ```python
  import random

  random.random()

  pets = ["cat", "dog", "fish"]
  # a random element from a sequence
  random.choice(pets)
  # shuffle a list (in place)
  random.shuffle(pets)

  # a random integer from 1 to 10 (inclusive)
  random.randint(1, 10)
  ```

CSV library

-
  ```
  import csv

  with open("numbers.csv") as f:
      r = csv.reader(f)
      for row in r:
          print row
  ```

Non-standard libraries

- We often might need libraries beyond standard libraries
- Anyone can publish their modules as python libraries
- PyPI - Python Package Index has all third-party libraries
- We can use `pip` to install the required packages

```
pip install numpy
```

- To get specific version you can add version to install command:

```
pip install numpy==1.16.3
```

Note: Learn about virtual environments to keep your dependencies clean

Numpy Library

- *“NumPy is the fundamental package for scientific computing with Python. It contains among other things:*
  - *a powerful N-dimensional array object*
  - *sophisticated (broadcasting) functions*
  - *tools for integrating C/C++ and Fortran code*
  - *useful linear algebra, Fourier transform, and random number capabilities”*

- Reference: https://www.numpy.org/

Numpy - Usage

- `import numpy as np`
- You can create arrays in many ways:
  - `a = np.array([2,3,4])`
  - `a = np.zeros([3,3])`
  - `a = np.arange(15).reshape(3,5)`
- Shape property gives shape (or dimensions) of the array
- We can perform array-wise operations
  - `a = 3 * a`

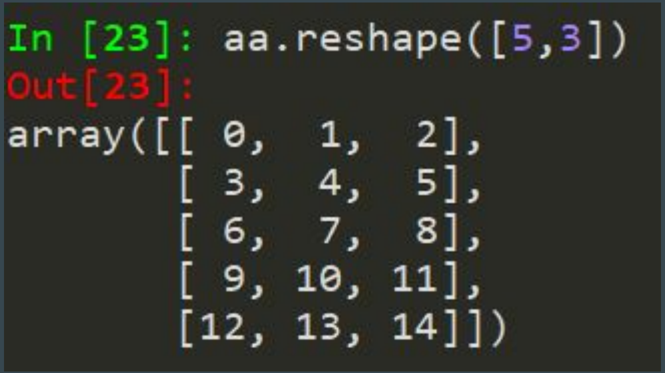
Numpy Indexing and Slicing

- Indexing and slicing is similar to list except we should be careful about the dimensions

  -
    ```
    aa = np.arange(15).reshape(3,5)
    print(aa[0,0])
    print(aa[0])
    print(aa[0, :])
    print(aa[:,2])
    ```

Numpy - Shape Manipulation

- We can shape the arrays if the requested shape still contains the same amount of elements. For instance, we cannot reshape a (3,5) array into (5,2)



```
In [23]: aa.reshape([5,3])
Out[23]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

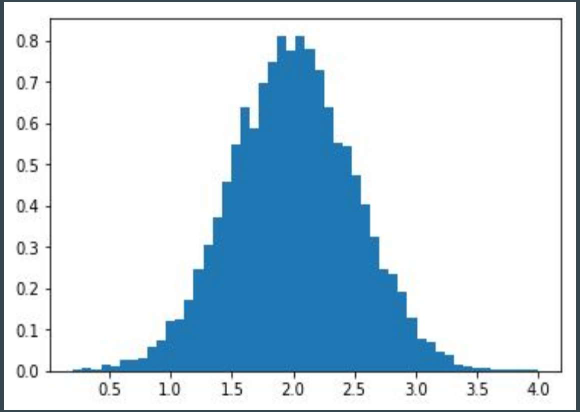
Numpy - Combining arrays

- vstack and hstack are used to combine two arrays along vertical and horizontal axis respectively

  -
    ```
    a = np.floor(10*np.random.random([2,3]))
    b = np.floor(10*np.random.random([2,3]))
    print(np.vstack((a,b)))
    print(np.hstack((a,b)))
    ```

Numpy + Plot

```
import numpy as np
import matplotlib.pyplot as plt
# Build a vector of 10000 normal deviates with
#variance 0.5^2 and mean 2
mu, sigma = 2, 0.5
v = np.random.normal(mu,sigma,10000)
# Plot a normalized histogram with 50 bins
plt.hist(v, bins=50, density=1)
# matplotlib version (plot)
plt.show()
```



Questions and Recap