Python

Why Python?

- Easy to use and general-purpose language
- Many scientific libraries for data analysis
- Many libraries for accessing data
- Free & open source
- Your company might already use it for sth else

Variables in Python

- Untyped variable
- Can be re-assigned (no final / val)
- Check current type with `type(variable)`

Data types

- Simple data types: strings, integers, floating point numbers, boolean

Data types

- Simple data types: strings, integers, floating point numbers, boolean

In [83]:
```
string1 = 'A string in single quotes allows embedded "double" quotes.'
string2 = "A string in double qoutes allows embedded 'single' quotes."
string3 = """A string in tripled quotes allows multi line string
with "double" and 'single' quotes."""
int1 = 99999999999999999999999999999999999999999999999 # unlimited precision in
 Python 3
float1 = 0.123456789
bool1 = True
string4 = str(int1) # type conversion
type(int1)
```

Out[83]:
```
int
```

List

List

In [65]:
```
shopping_list = [ 'milk', 'cheese', 'bread' ]
shopping_list.append(0) # add an element
shopping_list[0] # get first element
shopping_list[-1] # get last element
shopping_list[0:2] # get slice, left including, right excluding
len(shopping_list) # get lengh of a list
```

Out[65]:
```
4
```

Dict

Dict

In [66]:
```
d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]} # variant 1
d2 = dict(a='some value', b=[1, 2, 3, 4]) # variant2
d1['c'] = False # add an item
d1['a'] # get a value, KeyError if key does not exist
d1.get('x', 'default value') # avoid KeyError, get default value if key does not
 exist
```

Out[66]:
```
'default value'
```

Control Structures and Indentation

- Blocks are structured by colon and indentation

Control Structures and Indentation

- Blocks are structured by colon and indentation

In [67]:
```
shopping_list = [ 'milk', 'cheese', 'bread' ]
if not shopping_list:
    print('Nothing to buy today')
for item in shopping_list:
    print(f'Buy {item}')
```

```
Buy milk
Buy cheese
Buy bread
```

Functions & Methods

- Positional arguments, keyword arguments, default values
- Multiple return values (tulple)

Functions & Methods

- Positional arguments, keyword arguments, default values
- Multiple return values (tulple)

In [1]:
```
def add_and_multiply(a, b, c=0):
    """
    Add and multiply some numbers together
    """
    return a + b + c, a * b * c


result1 = add_and_multiply(1, 2) # use positional arguments
result1 # is a tuple
sum1, product1 = result1 # unpack the tuple

sum2, product2 = add_and_multiply(c=4, a=1, b=2) # use keyword arguments and unp
ack the result tuple
product2
```

Out[1]:
```
8
```

Classes, Objects, Constructor

Classes, Objects, Constructor

In [6]:
```
class C(object):
    def __init__(self, a=0, b=0):
        self.a = a
        self.b = b

    def get_sum(self):
        return self.a + self.b

c1 = C()
c2 = C(3,5)
c2.get_sum()
```

Out[6]:
```
8
```

Imports

- Non built-in modules must be imported
- Either module or single function/class or all

Imports

- Non built-in modules must be imported
- Either module or single function/class or all

In [77]:
```
import math # import math module
from random import random # import only the random function from the random modu
le
from datetime import * # import all classes from datetime module (avoid)

math.pi
timedelta(seconds=5)
random()
```

Out[77]:
```
0.6991655016949093
```

Python Code Completion and Help in Jupyter

- Code completion: `Tab`
- Python docstring: `Shift+Tab` (repeated)
- Help
  - `?object`: docstring of the class or function
  - `??object`: source code of module or class or function
- Doesn't work so well for built-ins
- h overview of Jupyter short cuts

Exercise 2 - Python

- Goals:
  - Remember your Python skills
  - Getting used to write Python code in Jupyter
- Tasks:
  - Try code completion and help
    - `Tab`, `Shift+Tab`, ?python module or class or function
  - Strings
    - Concatenate two strings,
    - Concatenate a string and a number
  - list
    - Concatenate two lists
    - Remove the second element from the list
  - dict
    - Change a value in a existing dict
    - Get a list of all keys and a list of all values of a dict

Libraries for Data Analysis

Numpy

- Fast and efficient N-dimensional array implementation
- Used as container for pasing data betweend algorithms and libraries
- Functions for working with large arrays and matrices, linear algebra operations
- Focused on numerical data, low-level numerics
- Homepage:http://www.numpy.org (http://www.numpy.org)

Pandas

- Build on top of numpy, adds functions for working with business data
- Data structues and tools for data analysis (in-memory)
- Tabluar data and time series
- Convenience functions for data import/export, plotting and join/merge of datasets
- Homepage: https://pandas.pydata.org/ (https://pandas.pydata.org/)
- Documentation: https://pandas.pydata.org/pandas-docs/stable/ (https://pandas.pydata.org/pandas-docs/stable/)

SciPy

- Collection of packages for different mathematical standard problems
  - stats: Probability distributions, various statistical tests, descriptive statistics
  - signal: Signal processing tools
  - linalg: Linear algebra routines and matrix decompositions
  - integrate: Numerical integration routines and differential equation solvers
- Homepage: https://www.scipy.org (https://www.scipy.org)

Visualization Libraries

- matplotlib (https://matplotlib.org (https://matplotlib.org))
  - Most popular visualization library in Python
  - Integrated in Pandas
- seaborn (https://seaborn.pydata.org (https://seaborn.pydata.org))
  - Based on matplotlib
  - Goal: Making prettier graphs easier
- bokeh (https://bokeh.pydata.org (https://bokeh.pydata.org))
  - Independent of matplotlib
  - Interactive graphics

Default imports

- Aliases `np`, `pd`, `plt` are very common.
- `%matplotlib inline` tells Jupyter to render plots inline
- `plt.rcParams["figure.figsize"] = [10,4]` makes the plots a bit larger

Default imports

- Aliases `np`, `pd`, `plt` are very common.
- `%matplotlib inline` tells Jupyter to render plots inline
- `plt.rcParams["figure.figsize"] = [10,4]` makes the plots a bit larger

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = [10,4]
```

NumPy

ndarray

- N-dimensional array object
- Main data structure in NumPy
- Many operations (matrix operations, linear algebra)

NumPy data types

NumPy dtype	Python type	Usage
int64/int32/int16	int	Integer numbers (overflow!)
float64/float32	float	Floating point numbers
complex128	complex	Complex numbers with real and imaginary components
bool	bool	True/False values
object	str	Text

NumPy data types

NumPy dtype	Python type	Usage
int64/int32/int16	int	Integer numbers (overflow!)
float64/float32	float	Floating point numbers
complex128	complex	Complex numbers with real and imaginary components
bool	bool	True/False values
object	str	Text

In [526]:
```
# Python 3: unlimited precision
17**17
```

Out[526]:
```
827240261886336764177
```

NumPy data types

NumPy dtype	Python type	Usage
int64/int32/int16	int	Integer numbers (overflow!)
float64/float32	float	Floating point numbers
complex128	complex	Complex numbers with real and imaginary components
bool	bool	True/False values
object	str	Text

In [526]:
```
# Python 3: unlimited precision
17**17
```

Out[526]:
```
827240261886336764177
```

In [528]:
```
# NumPy: Integer overflow
np.int64(17)**np.int64(17)
```

Out[528]:
```
-2863221430593058543
```

Array creation: from python array

Array creation: from python array

In [11]:
```
np.array([1,3,5,7])
```

Out[11]:
```
array([1, 3, 5, 7])
```

Array creation: from python array

In [11]:
```
np.array([1,3,5,7])
```

Out[11]:
```
array([1, 3, 5, 7])
```

In [404]:
```
np.array([[1,3,5,7],[2,4,6,8]])
```

Out[404]:
```
array([[1, 3, 5, 7],
       [2, 4, 6, 8]])
```

Array creation: reading from file

Array creation: reading from file

In [413]:
```
np.fromstring('1.1, 2.2, 3.3, 4.4, 5.5', sep=',')
```

Out[413]:
```
array([1.1, 2.2, 3.3, 4.4, 5.5])
```

Array creation: reading from file

In [413]:
```
np.fromstring('1.1, 2.2, 3.3, 4.4, 5.5', sep=',')
```

Out[413]:
```
array([1.1, 2.2, 3.3, 4.4, 5.5])
```

In [22]:
```
# skip the header row
# only read numeric columns: temperature, sold_icecream, sold_cups_coffee, sold_
coke
cafe = np.genfromtxt('data/cafe.csv', delimiter=',', skip_header=1, usecols=[1,2
,3,4], dtype=int)
cafe[0:10]
```

Out[22]:
```
array([[28, 40, 57, 44],
       [25, 36, 61, 19],
       [31, 45, 53, 15],
       [31, 47, 52, 26],
       [29, 45, 50, 23],
       [29, 44, 55, 42],
       [28, 42, 56, 22],
       [27, 40, 58, 31],
       [22, 32, 63, 26],
       [24, 35, 61, 19]])
```

Array creation: ranges

Array creation: ranges

In [415]:
```
# array range: start=0, stop, step=1
np.arange(10)
```

Out[415]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Array creation: ranges

In [415]:
```
# array range: start=0, stop, step=1
np.arange(10)
```

Out[415]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [532]:
```
# array range: start, stop, step, then reshape
np.arange(start=10, stop=30, step=2).reshape(2,5)
```

Out[532]:
```
array([[10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28]])
```

Get information about an ndarray

Get information about an ndarray

In [14]:
```
cafe.shape
```

Out[14]:
```
(35, 4)
```

In [17]:
```
cafe.size
```

Out[17]:
```
140
```

Get information about an ndarray

In [14]:
```
cafe.shape
```

Out[14]:
```
(35, 4)
```

In [17]:
```
cafe.size
```

Out[17]:
```
140
```

In [439]:
```
cafe.dtype
```

Out[439]:
```
dtype('int64')
```

Array indexing

Array indexing

In [488]:
```
# select first row
cafe[0]
```

Out[488]:
```
array([28, 40, 57, 44])
```

Array indexing

In [488]:
```
# select first row
cafe[0]
```

Out[488]:
```
array([28, 40, 57, 44])
```

In [489]:
```
# select range of rows
cafe[0:3]
```

Out[489]:
```
array([[28, 40, 57, 44],
       [25, 36, 61, 19],
       [31, 45, 53, 15]])
```

Array indexing

In [488]:
```
# select first row
cafe[0]
```

Out[488]:
```
array([28, 40, 57, 44])
```

In [489]:
```
# select range of rows
cafe[0:3]
```

Out[489]:
```
array([[28, 40, 57, 44],
       [25, 36, 61, 19],
       [31, 45, 53, 15]])
```

In [491]:
```
# select first column (temperatures)
cafe[:,0]
```

Out[491]:
```
array([28, 25, 31, 31, 29, 29, 28, 27, 22, 24, 26, 28, 30, 28, 29, 29, 28,
       31, 28, 24, 26, 25, 24, 25, 29, 30, 31, 30, 28, 28, 27, 30, 29, 27,
       26])
```

Array operations (1)

- Arithmetic operators on arrays apply elementwise

Array operations (1)

- Arithmetic operators on arrays apply elementwise

In [473]:
```
cafe[0:3]
```

Out[473]:
```
array([[28, 40, 57, 44],
       [25, 36, 61, 19],
       [31, 45, 53, 15]])
```

Array operations (1)

- Arithmetic operators on arrays apply elementwise

In [473]:
```
cafe[0:3]
```

Out[473]:
```
array([[28, 40, 57, 44],
       [25, 36, 61, 19],
       [31, 45, 53, 15]])
```

In [475]:
```
cafe_2 = cafe * 2
cafe_2[0:3]
```

Out[475]:
```
array([[ 56,  80, 114,  88],
       [ 50,  72, 122,  38],
       [ 62,  90, 106,  30]])
```

Array operations (1)

- Arithmetic operators on arrays apply elementwise

In [473]:
```
cafe[0:3]
```

Out[473]:
```
array([[28, 40, 57, 44],
       [25, 36, 61, 19],
       [31, 45, 53, 15]])
```

In [475]:
```
cafe_2 = cafe * 2
cafe_2[0:3]
```

Out[475]:
```
array([[ 56,  80, 114,  88],
       [ 50,  72, 122,  38],
       [ 62,  90, 106,  30]])
```

In [477]:
```
# convert temerature from Celcius to Fahrenheit
cafe_fahrenheit = cafe.copy()
cafe_fahrenheit[:,0] = cafe_fahrenheit[:,0] * 1.8 + 32
cafe_fahrenheit[0:3]
```

Out[477]:
```
array([[82, 40, 57, 44],
       [77, 36, 61, 19],
       [87, 45, 53, 15]])
```

Array operations (2)

- Summaries and basic statistics
- Axis: 0=rows, 1=columns

Array operations (2)

- Summaries and basic statistics
- Axis: 0=rows, 1=columns

In [513]:
```
# sum of sold ice creams column
cafe[:,1].sum()
```

Out[513]:
```
1456
```

Array operations (2)

- Summaries and basic statistics
- Axis: 0=rows, 1=columns

In [513]:
```
# sum of sold ice creams column
cafe[:,1].sum()
```

Out[513]:
```
1456
```

In [515]:
```
# sum of each column
cafe.sum(axis=0)
```

Out[515]:
```
array([ 970, 1456, 1984, 1057])
```

Array operations (2)

- Summaries and basic statistics
- Axis: 0=rows, 1=columns

In [513]:
```
# sum of sold ice creams column
cafe[:,1].sum()
```

Out[513]:
```
1456
```

In [515]:
```
# sum of each column
cafe.sum(axis=0)
```

Out[515]:
```
array([ 970, 1456, 1984, 1057])
```

In [518]:
```
# max/min of earch column
cafe.max(axis=0)
```

Out[518]:
```
array([31, 48, 63, 45])
```

Array operations (2)

- Summaries and basic statistics
- Axis: 0=rows, 1=columns

In [513]:
```
# sum of sold ice creams column
cafe[:,1].sum()
```

Out[513]:
```
1456
```

In [515]:
```
# sum of each column
cafe.sum(axis=0)
```

Out[515]:
```
array([ 970, 1456, 1984, 1057])
```

In [518]:
```
# max/min of earch column
cafe.max(axis=0)
```

Out[518]:
```
array([31, 48, 63, 45])
```

In [517]:
```
# mean/standard derivation/variance of each column
cafe.mean(axis=0) # std, var
```

Out[517]:
```
array([27.71428571, 41.6       , 56.68571429, 30.2       ])
```

Exercise 3 - NumPy

- Goals:
  - Library check
  - Getting used to NumPy arrays
- Tasks:
  - Check if the import statements for numpy, pandas, seaborn and matplotlib work on your machine
  - Create an one-dimensional and a two-dimensional ndarray
    - Use different data types (float, int)
    - Use different ways to create the arrays (python array, arange, etc.)
  - Print the arrays, shape, and dtype information
  - Apply some arithmetic operations to the arrays (square)
  - Calculate sum and mean for
    - the whole one-dimensional array
    - each column of the two-dimensional array