Phyton For Data Science

Cheat-Sheet Phyton Basic

BecomingHuman.AI

DataCamp

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

```
>>> x+2
7                    Sum of two variables
>>> x-2
3                    Subtraction of two variables
>>> x*2
10                   Multiplication of two variables
>>> x**2
25                   Exponentiation of a variable
>>> x%2
1                    Remainder of a variable
>>> x/float(2)
2.5                  Division of a variable
```

Calculations With Variables

str()	'5', '3.45', 'True'	Variables to strings
int()	5, 3, 1	Variables to integers
float()	5.0, 1.0	Variables to floats
bool()	True, True, True	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1]          Select item at index 1
>>> my_list[-3]         Select 3rd last item
```

Slice

```
>>> my_list[1:3]        Select items at index 1 and 2
>>> my_list[1:]         Select items after index 0
>>> my_list[:3]         Select items before index 3
>>> my_list[:]          Copy my_list
```

Subset Lists of Lists

```
>>> my_list2[1][0]      my_list[list][itemOfList]
>>> my_list2[1][:2]
```

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a)        Get the index of an item
>>> my_list.count(a)        Count an item
>>> my_list.append('!')     Append an item at a time
>>> my_list.remove('!')     Remove an item
>>> del(my_list[0:1])       Remove an item
>>> my_list.reverse()       Reverse the list
>>> my_list.extend('!')     Append an item
>>> my_list.pop(-1)         Remove an item
>>> my_list.insert(0,'!')   Insert an item
>>> my_list.sort()          Sort the list
```

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray =
np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1]         Select item at index 1
2
```

Slice

```
>>> my_array[0:2]       Select items at index 0 and 1
array([1, 2])
```

Subset 2D Numpy arrays

```
>>> my_2darray[:,0]     my_2darray[rows, columns]
array([1, 4])
```

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False, True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Operations

```
>>> my_array.shape          Get the dimensions of the array
>>> np.append(other_array)  Append items to an array
>>> np.insert(my_array, 1, 5)  Insert items in an array
>>> np.delete(my_array,[1])    Delete items in an array
>>> np.mean(my_array)       Mean of the array
>>> np.median(my_array)     Median of the array
>>> my_array.corrcoef()     Correlation coefficient
>>> np.std(my_array)        Standard deviation
```

Strings

Also see NumPy Arrays

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper()              String to uppercase
>>> my_string.lower()              String to lowercase
>>> my_string.count('w')           Count String elements
>>> my_string.replace('e', 'i')    Replace String elements
>>> my_string.strip()              Strip whitespaces
```

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

Install Python

ANACONDA. — Leading open data science platform powered by Pytho

SPYDER — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

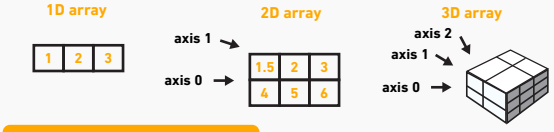
NumPy Basics Cheat Sheet

BecomingHuman.AI

DataCamp

NumPy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                    Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)    Create an array of ones
>>> d = np.arange(10,25,5)             Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9)                 Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7)               Create a constant array
>>> f = np.eye(2)                      Create a 2X2 identity matrix
>>> np.random.random((2,2))            Create an array with random values
>>> np.empty((3,2))                    Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=",")
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Data Types

```
>>> np.int64         Signed 64-bit integer types
>>> np.float32       Standard double-precision floating point
>>> np.complex       Complex numbers represented by 128 floats
>>> np.bool          Boolean type storing TRUE and FALSE
>>> np.object        Python object type values
>>> np.string_       Fixed-length string type
>>> np.unicode_      Fixed-length unicode type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                          Subtraction
  array([[-0.5, 0. , 0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                   Subtraction
>>> b + a                              Addition
  array([[ 2.5, 4. , 6. ],
         [ 5. , 7. , 9. ]])
>>> np.add(b,a)                        Addition
>>> a / b                              Division
  array([[ 0.66666667, 1. , 1. ],
         [ 0.25 , 0.4 , 0.5 ]])
>>> np.divide(a,b)                     Division
>>> a * b                              Multiplication
  array([[ 1.5, 4. , 9. ],
         [ 4. , 10. , 18. ]])
>>> np.multiply(a,b)                   Multiplication
>>> np.exp(b)                          Exponentiation
>>> np.sqrt(b)                         Square root
>>> np.sin(a)                          Print sines of an array
>>> np.cos(b)                          Element-wise cosine
>>> np.log(a)                          Element-wise natural logarithm
>>> e.dot(f)                           Dot product
  array([[ 7., 7.],
         [ 7., 7.]])
```

Comparison

```
>>> a == b                             Element-wise comparison
  array([[False, True, True],
         [False, False, False]], dtype=bool)
>>> a < 2                              Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)               Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()                            Array-wise sum
>>> a.min()                            Array-wise minimum value
>>> b.max(axis=0)                      Maximum value of an array row
>>> b.cumsum(axis=1)                   Cumulative sum of the elements
>>> a.mean()                           Mean
>>> b.median()                         Median
```

Copying Arrays

```
>>> h = a.view()          Create a view of the array with the same data
>>> np.copy(a)            Create a copy of the array
>>> h = a.copy()          Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()              Sort an array
>>> c.sort(axis=0)        Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2]                  Select the element at the 2nd index
  3
>>> b[1,2]                Select the element at row 1 column 2 (equivalent to b[1][2])
  6.0
```

Slicing

```
>>> a[0:2]                Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]              Select items at rows 0 and 1 in column 1
  array([ 2., 5.])
>>> b[:1]                 Select all items at row 0 (equivalent to b[0:1, :])
  array([[1.5, 2., 3.]])
>>> c[1,...]              Same as [1,:,:]
  array([[[ 3., 2., 1.],
          [ 4., 5., 6.]]])
>>> a[ : :-1]             Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]                Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]   Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]   Select a subset of the matrix's rows and columns
  array([[ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5],
         [ 4. , 5. , 6. , 4. ],
         [ 1.5, 2. , 3. , 1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)   Permute array dimensions
>>> i.T                   Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()             Flatten the array
>>> g.reshape(3,-2)       Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))       Return a new array with shape (2,6)
>>> np.append(h,g)         Append items to an array
>>> np.insert(a, 1, 5)     Insert items in an array
>>> np.delete(a,[1])       Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)   Concatenate arrays
  array([ 1, 2, 3, 10, 15, 20])
>>> np.vstack((a,b))               Stack arrays vertically (row-wise)
  array([[ 1. , 2. , 3. ],
         [ 1.5, 2. , 3. ],
         [ 4. , 5. , 6. ]])
>>> np.r_[e,f]                     Stack arrays vertically (row-wise)
>>> np.hstack((e,f))               Stack arrays horizontally (column-wise)
  array([[ 7., 7., 1., 0.],
         [ 7., 7., 0., 1.]])
>>> np.column_stack((a,d))         Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                     Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)                 Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])] index
>>> np.vsplit(c,2) Split the array vertically at the 2nd index
  [array([[[ 1.5, 2. , 1. ],
           [ 4. , 5. , 6. ]]]),
```

Pandas Basics

Cheat Sheet

BecomingHuman.AI

DataCamp

Use the following import convention: >>> import pandas as pd

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.

Pandas Data Structures

Series

A one-dimensional labeled array a capable of holding any data type

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Data Frame

A two-dimensional labeled data structure with columns of potentially different types

	Belgium	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

a	3
b	-5
c	7
d	4

Dropping

```
>>> s.drop(['a', 'c'])
```
Drop values from rows (axis=0)

```
>>> df.drop('Country', axis=1)
```
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index()
```
Sort by labels along an axis

```
>>> df.sort_values(by='Country')
```
Sort by the values along an axis

```
>>> df.rank()
```
Assign ranks to entries

Retrieving Series/ DataFrame Information

```
>>> df.shape
```
(rows,columns)

```
>>> df.index
```
Describe index

```
>>> df.columns
```
Describe DataFrame columns

```
>>> df.info()
```
Info on DataFrame

```
>>> df.count()
```
Number of non-NA values

Summary

```
>>> df.sum()
```
Sum of values

```
>>> df.cumsum()
```
Cummulative sum of values

```
>>> df.min()/df.max()
```
Minimum/maximum values

```
>>> df.idxmin()/df.idxmax()
```
Minimum/Maximum index value

```
>>> df.describe()
```
Summary statistics

```
>>> df.mean()
```
Mean of values

```
>>> df.median()
```
Median of values

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
-5
```
Get one element

```
>>> df[1:]
   Country Capital      Population
1  India   New Delhi    1303171035
2  Brazil  Brasilia     207847528
```
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.iat([0],[0])
'Belgium'
```
Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at([0], ['Country']) 'Belgium'
```
Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital      Brasilia
Population   207847528
```
Select single row of subset of rows

```
>>> df.ix[:,'Capital']
0 Brussels
1 New Delhi
2 Brasilia
```
Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
'New Delhi'
```
Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
```
Series s where value is not >1

```
>>> s[(s < -1) | (s > 2)]
```
s where value is <-1 or >2

```
>>> df[df['Population']>1200000000]
```
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```
Set index a of Series s to 6

Asking For Help

```
>>> help(pd.Series.loc)
```

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
```
Apply function

```
>>> df.applymap(f)
```
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a 10.0
  b NaN
  c 5.0
  d 7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a 10.0
  b -5.0
  c 5.0
  d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql()is **a convenience wrapper around** read_sql_table() and read_sql_query()

```
>>> pd.to_sql('myDf', engine)
```

Pandas Cheat Sheet

BecomingHuman.AI

Pandas Data Structures

Pivot

```
>>> df3= df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

Spread rows into columns

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

Type	a	b	c
Date			
2016-03-01	11.432	NaN	20.784
2016-03-02	1.303	13.031	NaN
2016-03-03	99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,
                         values='Value',
                         index='Date',
                         columns='Type'])
```

Spread rows into columns

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

Melt

```
>>> pd.melt(df2,
            id_vars=['Date'],
            value_vars=['Type', 'Value'],
            value_name='Observations')
```

Gather columns into rows

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()]      Select cols with any vals >1
>>> df3.loc[:,(df3>1).all()]      Select cols with vals > 1
>>> df3.loc[:,df3.isnull().any()] Select cols with NaN
>>> df3.loc[:,df3.notnull().all()] Select cols without NaN
```

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))] Find same elements
>>> df3.filter(items="a","b"])      Filter on values
>>> df.select(lambda x: not x%5)    Select specific elements
```

Where

```
>>> s.where(s > 0)                  Subset the data
```

Query

```
>>> df6.query('second > first')     Query DataFrame
```

Setting/Resetting Index

```
>>> df.set_index('Country')         Set the index
>>> df4 = df.reset_index()          Reset the index
>>> df = df.rename(index=str,       Rename DataFrame
                   columns={"Country":"cntry",
                            "Capital":"cptl",
                            "Population":"ppltn"})
```

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528
3	Brazil	Brasília	207847528

Forward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

0	3
1	3
2	3
3	3
4	3

MultiIndexing

```
>>> arrays = [np.array([1,2,3]),
              np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(["Date", "Type"])
```

Duplicate Data

```
>>> s3.unique()                     Return unique values
>>> df2.duplicated('Type')          Check duplicates
>>> df2.drop_duplicates('Type', keep='last') Drop duplicates
>>> df.index.duplicated()           Drop duplicates
```

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x), 'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

```
>>> df.dropna()                     Drop NaN value
>>> df3.fillna(df3.mean())          Fill NaN values with a predetermined value
>>> df2.replace("a", "f")           Replace values with others
```

Combining Data

data1

	X1	X2
a	11.432
b	1.303
c	99.906

data2

	X1	X3
a	20.784
b	NaN
d	20.784

Pivot

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

	X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN
d	NaN	

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

```
>>> s.append(s2)
```

Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1, keys=['One','Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

Dates

```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1', periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

Visualization

```
>>> import matplotlib.pyplot as plt
```

```
>>> s.plot()
>>> plt.show()
```

```
>>> df2.plot()
>>> plt.show()
```