PythonCheatSheet

April 11, 2022

1 Python Cheatsheet

1.1 Contents

To run a cell, press **Shift+Enter** or click **Run** at the top of the page.

1.2 1. Syntax and whitespace

Python uses indented space to indicate the level of statements. The following cell is an example where '**if**' and '**else**' are in same level, while '**print**' is separated by space to a different level. Spacing should be the same for items that are on the same level.

[ ]:
```
student_number = input("Enter your student number:")
if int(student_number) != 0:
    print("Welcome student {}".format(student_number))
else:
    print("Try again!")
```

```
Enter your student number:5
Welcome student 5
```

1.3 2. Comments

In Python, comments start with hash '#' and extend to the end of the line. '#' can be at the begining of the line or after code.

[ ]:
```
# This is code to print hello world!

print("Hello world!") # Print statement for hello world
print("# is not a comment in this case")
```

```
Hello world!
# is not a comment in this case
```

1.4 3. Numbers and operations

Like with other programming languages, there are four types of numbers: - Integers (e.g., 1, 20, 45, 1000) indicated by *int* - Floating point numbers (e.g., 1.25, 20.35, 1000.00) indicated by *float* - Long integers - Complex numbers (e.g., x+2y where x is known)

Operation	Result
x + y	Sum of x and y
x - y	Difference of x and y
x * y	Product of x and y
x / y	Quotient of x and y
x // y	Quotient of x and y (floored)
x % y	Remainder of x / y
abs(x)	Absolute value of x
int(x)	x converted to integer
long(x)	x converted to long integer
float(x)	x converted to floating point
pow(x, y)	x to the power y
x ** y	x to the power y

[ ]:
```
# Number examples
a = 5 + 8
print("Sum of int numbers: {} and number format is {}".format(a, type(a)))

b = 5 + 2.3
print ("Sum of int and {} and number format is {}".format(b, type(b)))
```

```
Sum of int numbers: 13 and number format is <class 'int'>
Sum of int and 7.3 and number format is <class 'float'>
```

1.5 4. String manipulation

Python has rich features like other programming languages for string manipulation.

[ ]:
```
# Store strings in a variable
test_word = "hello world to everyone"

# Print the test_word value
print(test_word)

# Use [] to access the character of the string. The first character is␣
 ↪indicated by '0'.
print(test_word[0])

# Use the len() function to find the length of the string
print(len(test_word))

# Some examples of finding in strings
print(test_word.count('l')) # Count number of times l repeats in the string
print(test_word.find("o")) # Find letter 'o' in the string. Returns the␣
 ↪position of first match.
print(test_word.count(' ')) # Count number of spaces in the string
print(test_word.upper()) # Change the string to uppercase
print(test_word.lower()) # Change the string to lowercase
print(test_word.replace("everyone","you")) # Replace word "everyone" with "you"
print(test_word.title()) # Change string to title format
print(test_word + "!!!") # Concatenate strings
print(":".join(test_word)) # Add ":" between each character
print("".join(reversed(test_word))) # Reverse the string
```

```
hello world to everyone
h
23
3
4
3
HELLO WORLD TO EVERYONE
hello world to everyone
hello world to you
Hello World To Everyone
hello world to everyone!!!
h:e:l:l:o: :w:o:r:l:d: :t:o: :e:v:e:r:y:o:n:e
enoyreve ot dlrow olleh
```

1.6 5. Lists, tuples, and dictionaries

Python supports data types lists, tuples, dictionaries, and arrays.

1.6.1 Lists

A list is created by placing all the items (elements) inside square brackets [ ] separated by commas. A list can have any number of items, and they may be of different types (integer, float, strings, etc.).

[ ]:
```
# A Python list is similar to an array. You can create an empty list too.

my_list = []

first_list = [3, 5, 7, 10]
second_list = [1, 'python', 3]
```

[ ]:
```
# Nest multiple lists
nested_list = [first_list, second_list]
nested_list
```

[ ]:
```
[[3, 5, 7, 10], [1, 'python', 3]]
```

[ ]:
```
# Combine multiple lists
combined_list = first_list + second_list
combined_list
```

[ ]:
```
[3, 5, 7, 10, 1, 'python', 3]
```

[ ]:
```
# You can slice a list, just like strings
combined_list[0:3]
```

[ ]:
```
[3, 5, 7]
```

[ ]:
```
# Append a new entry to the list
combined_list.append(600)
combined_list
```

[ ]:
```
[3, 5, 7, 10, 1, 'python', 3, 600]
```

[ ]:
```
# Remove the last entry from the list
combined_list.pop()
```

[ ]:
```
600
```

[ ]:
```
# Iterate the list
for item in combined_list:
    print(item)
```

```
3
5
7
```

```
10
1
python
3
```

1.6.2 Tuples

A tuple is similar to a list, but you use them with parentheses ( ) instead of square brackets. The main difference is that a tuple is immutable, while a list is mutable.

[ ]:
```
my_tuple = (1, 2, 3, 4, 5)
my_tuple[1:4]
```

[ ]:
```
(2, 3, 4)
```

1.6.3 Dictionaries

A dictionary is also known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

[ ]:
```
desk_location = {'jack': 123, 'joe': 234, 'hary': 543}
desk_location['jack']
```

[ ]:
```
123
```

1.7 6. JSON

JSON is text writen in JavaScript Object Notation. Python has a built-in package called `json` that can be used to work with JSON data.

[ ]:
```
import json

# Sample JSON data
x = '{"first_name":"Jane", "last_name":"Doe", "age":25, "city":"Chicago"}'

# Read JSON data
y = json.loads(x)

# Print the output, which is similar to a dictonary
print("Employee name is "+ y["first_name"] + " " + y["last_name"])
```

```
Employee name is Jane Doe
```

1.8 7. Loops

If, Else, ElIf loop: Python supports conditional statements like any other programming language. Python relies on indentation (whitespace at the begining of the line) to define the scope of the code.

[ ]:
```
a = 22
b = 33
c = 100

# if ... else example
if a > b:
    print("a is greater than b")
else:
    print("b is greater than a")


# if .. else .. elif example

if a > b:
    print("a is greater than b")
elif b > c:
    print("b is greater than c")
else:
    print("b is greater than a and c is greater than b")
```

```
b is greater than a
b is greater than a and c is greater than b
```

While loop: Runs a set of statements as long as the condition is true

[ ]:
```
# Sample while example
i = 1
while i < 10:
    print("count is " + str(i))
    i += 1

print("="*10)

# Continue to next iteration if x is 2. Finally, print message once the␣
 ↪condition is false.

x = 0
while x < 5:
    x += 1
    if x == 2:
        continue
    print(x)
else:
```

```
    print("x is no longer less than 5")
```

```
count is 1
count is 2
count is 3
count is 4
count is 5
count is 6
count is 7
count is 8
count is 9
==========
1
3
4
5
x is no longer less than 5
```

For loop: A `For` loop is more like an iterator in Python. A `For` loop is used for iterating over a sequence (list, tuple, dictionay, set, string, or range).

[ ]:
```
# Sample for loop examples
fruits = ["orange", "banana", "apple", "grape", "cherry"]
for fruit in fruits:
    print(fruit)

print("\n")
print("="*10)
print("\n")

# Iterating range
for x in range(1, 10, 2):
    print(x)
else:
    print("task complete")

print("\n")
print("="*10)
print("\n")

# Iterating multiple lists
traffic_lights = ["red", "yellow", "green"]
action = ["stop", "slow down", "go"]

for light in traffic_lights:
    for task in action:
        print(light, task)
```

```
orange
banana
apple
grape
cherry


==========


1
3
5
7
9
task complete


==========


red stop
red slow down
red go
yellow stop
yellow slow down
yellow go
green stop
green slow down
green go
```

1.9 8. File handling

The key function for working with files in Python is the `open()` function. The `open()` function takes two parameters: filename and mode.

There are four different methods (modes) for opening a file:

- ”r” - Read
- ”a” - Append
- ”w” - Write
- ”x” - Create

In addition, you can specify if the file should be handled in binary or text mode.

- ”t” - Text
- ”b” - Binary

[ ]:
```
# Let's create a test text file
!echo "This is a test file with text in it. This is the first line." > test.txt
!echo "This is the second line." >> test.txt
!echo "This is the third line." >> test.txt
```

[ ]:
```
# Read file
file = open('test.txt', 'r')
print(file.read())
file.close()

print("\n")
print("="*10)
print("\n")

# Read first 10 characters of the file
file = open('test.txt', 'r')
print(file.read(10))
file.close()

print("\n")
print("="*10)
print("\n")

# Read line from the file

file = open('test.txt', 'r')
print(file.readline())
file.close()
```

```
This is a test file with text in it. This is the first line.
This is the second line.
This is the third line.



==========


This is a


==========


This is a test file with text in it. This is the first line.
```

[ ]:
```
# Create new file

file = open('test2.txt', 'w')
file.write("This is content in the new test2 file.")
file.close()

# Read the content of the new file
file = open('test2.txt', 'r')
print(file.read())
file.close()
```

```
This is content in the new test2 file.
```

[ ]:
```
# Update file
file = open('test2.txt', 'a')
file.write("\nThis is additional content in the new file.")
file.close()

# Read the content of the new file
file = open('test2.txt', 'r')
print(file.read())
file.close()
```

```
This is content in the new test2 file.
This is additional content in the new file.
```

[ ]:
```
# Delete file
import os
file_names = ["test.txt", "test2.txt"]
for item in file_names:
    if os.path.exists(item):
        os.remove(item)
        print(f"File {item} removed successfully!")
    else:
        print(f"{item} file does not exist.")
```

```
File test.txt removed successfully!
File test2.txt removed successfully!
```

1.10 9. Functions

A function is a block of code that runs when it is called. You can pass data, or *parameters*, into the function. In Python, a function is defined by `def`.

[ ]:
```
# Defining a function
def new_funct():
    print("A simple function")
```

```
# Calling the function
new_funct()
```

```
A simple function
```

[ ]:
```
# Sample fuction with parameters

def param_funct(first_name):
    print(f"Employee name is {first_name}.")

param_funct("Harry")
param_funct("Larry")
param_funct("Shally")
```

```
Employee name is Harry.
Employee name is Larry.
Employee name is Shally.
```

Anonymous functions (lambda): A lambda is a small anonymous function. A lambda function can take any number of arguments but only one expression.

[ ]:
```
# Sample lambda example
x = lambda y: y + 100
print(x(15))

print("\n")
print("="*10)
print("\n")

x = lambda a, b: a*b/100
print(x(2,4))
```

```
115


==========


0.08
```

1.11 10. Working with datetime

A `datetime` module in Python can be used to work with date objects.

[ ]:
```
import datetime

x = datetime.datetime.now()
```

```
print(x)
print(x.year)
print(x.strftime("%A"))
print(x.strftime("%B"))
print(x.strftime("%d"))
print(x.strftime("%H:%M:%S %p"))
```

```
2022-03-10 20:18:59.907021
2022
Thursday
March
10
20:18:59 PM
```

1.12 11. NumPy

NumPy is the fundamental package for scientific computing with Python. Among other things, it contains:

- 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

[ ]:
```
# Install NumPy using pip
!pip install --upgrade pip
!pip install numpy
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.7/dist-packages
(22.0.4)
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-
packages (1.21.5)
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

[ ]:
```
# Import NumPy module
import numpy as np
```

1.12.1 Inspecting your array

[ ]:
```
# Create array
a = np.arange(15).reshape(3, 5) # Create array with range 0-14 in 3 by 5␣
 ↪dimension
b = np.zeros((3,5)) # Create array with zeroes
c = np.ones( (2,3,4), dtype=np.int16 ) # Createarray with ones and defining␣
 ↪data types
d = np.ones((3,5))
```

[ ]:
```
a.shape # Array dimension
```

[ ]:
```
(3, 5)
```

[ ]:
```
len(b)# Length of array
```

[ ]:
```
3
```

[ ]:
```
c.ndim # Number of array dimensions
```

[ ]:
```
3
```

[ ]:
```
a.size # Number of array elements
```

[ ]:
```
15
```

[ ]:
```
b.dtype # Data type of array elements
```

[ ]:
```
dtype('float64')
```

[ ]:
```
c.dtype.name # Name of data type
```

[ ]:
```
'int16'
```

[ ]:
```
c.astype(float) # Convert an array type to a different type
```

[ ]:
```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

1.12.2 Basic math operations

[ ]:
```
# Create array
a = np.arange(15).reshape(3, 5) # Create array with range 0-14 in 3 by 5␣
 ↪dimension
b = np.zeros((3,5)) # Create array with zeroes
c = np.ones( (2,3,4), dtype=np.int16 ) # Createarray with ones and defining␣
 ↪data types
d = np.ones((3,5))
```

[ ]:
```
np.add(a,b) # Addition
```

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

[ ]:
```
np.subtract(a,b) # Substraction
```

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

[ ]:
```
np.divide(a,d) # Division
```

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

[ ]:
```
np.multiply(a,d) # Multiplication
```

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

[ ]:
```
np.array_equal(a,b) # Comparison - arraywise
```

[ ]:
```
False
```

1.12.3 Aggregate functions

[ ]:
```
# Create array
a = np.arange(15).reshape(3, 5) # Create array with range 0-14 in 3 by 5␣
 ↪dimension
b = np.zeros((3,5)) # Create array with zeroes
c = np.ones( (2,3,4), dtype=np.int16 ) # Createarray with ones and defining␣
 ↪data types
```

```
d = np.ones((3,5))
```

[ ]:
```
a.sum() # Array-wise sum
```

[ ]:
```
105
```

[ ]:
```
a.min() # Array-wise min value
```

[ ]:
```
0
```

[ ]:
```
a.mean() # Array-wise mean
```

[ ]:
```
7.0
```

[ ]:
```
a.max(axis=0) # Max value of array row
```

[ ]:
```
array([10, 11, 12, 13, 14])
```

[ ]:
```
np.std(a) # Standard deviation
```

[ ]:
```
4.320493798938574
```

1.12.4 Subsetting, slicing, and indexing

[ ]:
```
# Create array
a = np.arange(15).reshape(3, 5) # Create array with range 0-14 in 3 by 5␣
 ↪dimension
b = np.zeros((3,5)) # Create array with zeroes
c = np.ones( (2,3,4), dtype=np.int16 ) # Createarray with ones and defining␣
 ↪data types
d = np.ones((3,5))
```

[ ]:
```
a[1,2] # Select element of row 1 and column 2
```

[ ]:
```
7
```

[ ]:
```
a[0:2] # Select items on index 0 and 1
```

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

[ ]:
```
a[:1] # Select all items at row 0
```

[ ]:
```
array([[0, 1, 2, 3, 4]])
```

[ ]:
```
a[-1:] # Select all items from last row
```

[ ]:
```
array([[10, 11, 12, 13, 14]])
```

[ ]:
```
a[a<2] # Select elements from 'a' that are less than 2
```

[ ]:
```
array([0, 1])
```

1.12.5 Array manipulation

[ ]:
```
# Create array
a = np.arange(15).reshape(3, 5) # Create array with range 0-14 in 3 by 5␣
 ↪dimension
b = np.zeros((3,5)) # Create array with zeroes
c = np.ones( (2,3,4), dtype=np.int16 ) # Createarray with ones and defining␣
 ↪data types
d = np.ones((3,5))
```

[ ]:
```
np.transpose(a) # Transpose array 'a'
```

[ ]:
```
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

[ ]:
```
a.ravel() # Flatten the array
```

[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

[ ]:
```
a.reshape(5,-2) # Reshape but don't change the data
```

[ ]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

[ ]:
```
np.append(a,b) # Append items to the array
```

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

[ ]:
```
np.concatenate((a,d), axis=0) # Concatenate arrays
```

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

```
        [10., 11., 12., 13., 14.],
        [ 1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.],
        [ 1.,  1.,  1.,  1.,  1.]])
```

[ ]:
```
np.vsplit(a,3) # Split array vertically at 3rd index
```

[ ]:
```
[array([[0, 1, 2, 3, 4]]),
 array([[5, 6, 7, 8, 9]]),
 array([[10, 11, 12, 13, 14]])]
```

[ ]:
```
np.hsplit(a,5) # Split array horizontally at 5th index
```

[ ]:
```
[array([[ 0],
        [ 5],
        [10]]), array([[ 1],
        [ 6],
        [11]]), array([[ 2],
        [ 7],
        [12]]), array([[ 3],
        [ 8],
        [13]]), array([[ 4],
        [ 9],
        [14]])]
```

1.13 Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas DataFrames are the most widely used in-memory representation of complex data collections within Python.

[ ]:
```
# Install pandas, xlrd, and openpyxl using pip
!pip install pandas
!pip install xlrd openpyxl
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
(1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-
packages (from pandas) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-
packages (from pandas) (1.21.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
```

```
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
Requirement already satisfied: xlrd in /usr/local/lib/python3.7/dist-
packages (1.1.0)
Requirement already satisfied: openpyxl in /usr/local/lib/python3.7/dist-
packages (3.0.9)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.7/dist-
packages (from openpyxl) (1.1.0)
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

[ ]:
```
# Import NumPy and Pandas modules
import numpy as np
import pandas as pd
```

[ ]:
```
# Sample dataframe df
df = pd.DataFrame({'num_legs': [2, 4, np.nan, 0],
                   'num_wings': [2, 0, 0, 0],
                   'num_specimen_seen': [10, np.nan, 1, 8]},
                   index=['falcon', 'dog', 'spider', 'fish'])
df # Display dataframe df
```

[ ]:

	num_legs	num_wings	num_specimen_seen
falcon	2.0	2	10.0
dog	4.0	0	NaN
spider	NaN	0	1.0
fish	0.0	0	8.0

[ ]:
```
# Another sample dataframe df1 - using NumPy array with datetime index and␣
 ↪labeled column
df1 = pd.date_range('20130101', periods=6)
df1 = pd.DataFrame(np.random.randn(6, 4), index=df1, columns=list('ABCD'))
df1 # Display dataframe df1
```

[ ]:

	A	B	C	D
2013-01-01	0.585722	-0.649741	-3.578389	1.256532
2013-01-02	0.365194	-0.030423	-1.903047	-1.967513
2013-01-03	-1.273320	-0.608034	-0.118732	-0.044090
2013-01-04	-1.352582	1.306480	0.064463	-0.156061
2013-01-05	1.675461	-0.836540	-0.209184	0.012749
2013-01-06	0.006478	-0.977767	1.213337	-0.512744

1.13.1 Viewing data

[ ]:
```
df1 = pd.date_range('20130101', periods=6)
df1 = pd.DataFrame(np.random.randn(6, 4), index=df1, columns=list('ABCD'))
```

[ ]:
```
df1.head(2) # View top data
```

[ ]:
```
                   A         B         C         D
2013-01-01  0.856162 -0.967504 -1.239942  0.992299
2013-01-02  2.456804  1.249084  0.580066 -0.827189
```

[ ]:
```
df1.tail(2) # View bottom data
```

[ ]:
```
                   A         B         C         D
2013-01-05  0.146800 -0.728447  1.299218  1.283088
2013-01-06 -0.358769 -0.986644 -0.193084 -0.282883
```

[ ]:
```
df1.index # Display index column
```

[ ]:
```
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

[ ]:
```
df1.dtypes # Inspect datatypes
```

[ ]:
```
A    float64
B    float64
C    float64
D    float64
dtype: object
```

[ ]:
```
df1.describe() # Display quick statistics summary of data
```

[ ]:
```
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.926104 -0.568749 -0.038979  0.496867
std    0.979895  0.936510  0.899033  0.846609
min   -0.358769 -1.418179 -1.239942 -0.827189
25%    0.324140 -0.981859 -0.552038 -0.010814
50%    1.002041 -0.847976 -0.100762  0.898845
75%    1.267759 -0.602716  0.432939  1.005947
max    2.456804  1.249084  1.299218  1.283088
```

1.13.2 Subsetting, slicing, and indexing

[ ]:
```
df1 = pd.date_range('20130101', periods=6)
df1 = pd.DataFrame(np.random.randn(6, 4), index=df1, columns=list('ABCD'))
```

[ ]:
```
df1.T # Transpose data
```

[ ]:
```
   2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
A    0.624376   -0.032064   -0.477969    2.134176    0.171814    0.060342
B   -0.688370   -0.467953    0.283626    0.980079   -0.335895    0.889149
C    0.246922   -1.080885    0.205450   -1.205140   -0.919308    0.602191
D    3.277770   -1.113337    1.864838    0.487757   -1.183843   -0.208107
```

[ ]:
```
df1.sort_index(axis=1, ascending=False) # Sort by an axis
```

[ ]:
```
                   D         C         B         A
2013-01-01  3.277770  0.246922 -0.688370  0.624376
2013-01-02 -1.113337 -1.080885 -0.467953 -0.032064
2013-01-03  1.864838  0.205450  0.283626 -0.477969
2013-01-04  0.487757 -1.205140  0.980079  2.134176
2013-01-05 -1.183843 -0.919308 -0.335895  0.171814
2013-01-06 -0.208107  0.602191  0.889149  0.060342
```

[ ]:
```
df1.sort_values(by='B') # Sort by values
```

[ ]:
```
                   A         B         C         D
2013-01-01  0.624376 -0.688370  0.246922  3.277770
2013-01-02 -0.032064 -0.467953 -1.080885 -1.113337
2013-01-05  0.171814 -0.335895 -0.919308 -1.183843
2013-01-03 -0.477969  0.283626  0.205450  1.864838
2013-01-06  0.060342  0.889149  0.602191 -0.208107
2013-01-04  2.134176  0.980079 -1.205140  0.487757
```

[ ]:
```
df1['A'] # Select column A
```

[ ]:
```
2013-01-01    0.624376
2013-01-02   -0.032064
2013-01-03   -0.477969
2013-01-04    2.134176
2013-01-05    0.171814
2013-01-06    0.060342
Freq: D, Name: A, dtype: float64
```

[ ]:
```
df1[0:3] # Select index 0 to 2
```

[ ]:
```
                   A         B         C         D
2013-01-01  0.624376 -0.688370  0.246922  3.277770
```

```
2013-01-02 -0.032064 -0.467953 -1.080885 -1.113337
2013-01-03 -0.477969  0.283626  0.205450  1.864838
```

[ ]:
```
df1['20130102':'20130104'] # Select from index matching the values
```

[ ]:
```
                   A         B         C         D
2013-01-02 -0.032064 -0.467953 -1.080885 -1.113337
2013-01-03 -0.477969  0.283626  0.205450  1.864838
2013-01-04  2.134176  0.980079 -1.205140  0.487757
```

[ ]:
```
df1.loc[:, ['A', 'B']] # Select on a multi-axis by label
```

[ ]:
```
                   A         B
2013-01-01  0.624376 -0.688370
2013-01-02 -0.032064 -0.467953
2013-01-03 -0.477969  0.283626
2013-01-04  2.134176  0.980079
2013-01-05  0.171814 -0.335895
2013-01-06  0.060342  0.889149
```

[ ]:
```
df1.iloc[3] # Select via the position of the passed integers
```

[ ]:
```
A    2.134176
B    0.980079
C   -1.205140
D    0.487757
Name: 2013-01-04 00:00:00, dtype: float64
```

[ ]:
```
df1[df1 > 0] # Select values from a DataFrame where a boolean condition is met
```

[ ]:
```
                   A         B         C         D
2013-01-01  0.624376       NaN  0.246922  3.277770
2013-01-02       NaN       NaN       NaN       NaN
2013-01-03       NaN  0.283626  0.205450  1.864838
2013-01-04  2.134176  0.980079       NaN  0.487757
2013-01-05  0.171814       NaN       NaN       NaN
2013-01-06  0.060342  0.889149  0.602191       NaN
```

[ ]:
```
df2 = df1.copy() # Copy the df1 dataset to df2
df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three'] # Add column E with␣
 ↪value
df2[df2['E'].isin(['two', 'four'])] # Use isin method for filtering
```

[ ]:
```
                   A         B         C         D     E
2013-01-03 -0.477969  0.283626  0.205450  1.864838   two
2013-01-05  0.171814 -0.335895 -0.919308 -1.183843  four
```

1.13.3 Missing data

Pandas primarily uses the value `np.nan` to represent missing data. It is not included in computations by default.

[ ]:
```
df = pd.DataFrame({'num_legs': [2, 4, np.nan, 0],
                   'num_wings': [2, 0, 0, 0],
                   'num_specimen_seen': [10, np.nan, 1, 8]},
                   index=['falcon', 'dog', 'spider', 'fish'])
```

[ ]:
```
df.dropna(how='any') # Drop any rows that have missing data
```

[ ]:
```
        num_legs  num_wings  num_specimen_seen
falcon       2.0          2               10.0
fish         0.0          0                8.0
```

[ ]:
```
df.dropna(how='any', axis=1) # Drop any columns that have missing data
```

[ ]:
```
        num_wings
falcon          2
dog             0
spider          0
fish            0
```

[ ]:
```
df.fillna(value=5) # Fill missing data with value 5
```

[ ]:
```
        num_legs  num_wings  num_specimen_seen
falcon       2.0          2               10.0
dog          4.0          0                5.0
spider       5.0          0                1.0
fish         0.0          0                8.0
```

[ ]:
```
pd.isna(df) # To get boolean mask where data is missing
```

[ ]:
```
        num_legs  num_wings  num_specimen_seen
falcon     False      False              False
dog        False      False               True
spider      True      False              False
fish       False      False              False
```

1.13.4 File handling

[ ]:
```
df = pd.DataFrame({'num_legs': [2, 4, np.nan, 0],
                   'num_wings': [2, 0, 0, 0],
                   'num_specimen_seen': [10, np.nan, 1, 8]},
                   index=['falcon', 'dog', 'spider', 'fish'])
```

[ ]:
```
df.to_csv('foo.csv') # Write to CSV file
```

[ ]:
```
pd.read_csv('foo.csv') # Read from CSV file
```

[ ]:

	Unnamed: 0	num_legs	num_wings	num_specimen_seen
0	falcon	2.0	2	10.0
1	dog	4.0	0	NaN
2	spider	NaN	0	1.0
3	fish	0.0	0	8.0

[ ]:
```
df.to_excel('foo.xlsx', sheet_name='Sheet1') # Write to Microsoft Excel file
```

[ ]:
```
pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'],␣
↪engine='openpyxl') # Read from Microsoft Excel file
```

[ ]:

	Unnamed: 0	num_legs	num_wings	num_specimen_seen
0	falcon	2.0	2	10.0
1	dog	4.0	0	NaN
2	spider	NaN	0	1.0
3	fish	0.0	0	8.0

1.13.5 Plotting

[ ]:
```
# Install Matplotlib using pip
!pip install matplotlib
```

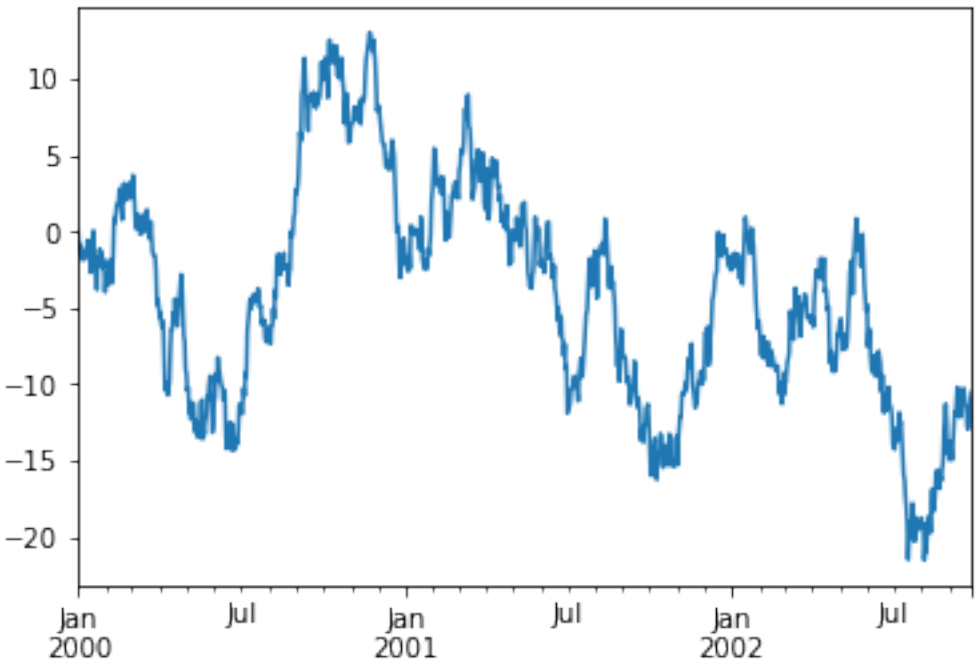
```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-
packages (3.2.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
packages (from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (3.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-
packages (from matplotlib) (1.21.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.1->matplotlib) (1.15.0)
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
```

[ ]:
```
from matplotlib import pyplot as plt # Import Matplotlib module
```

[ ]:
```
# Generate random time-series data
ts = pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2000',␣
↪periods=1000))
ts.head()
```

[ ]:
```
2000-01-01   -0.029855
2000-01-02   -1.234720
2000-01-03    0.406091
2000-01-04   -0.719265
2000-01-05    0.443042
Freq: D, dtype: float64
```

[ ]:
```
ts = ts.cumsum()
ts.plot() # Plot graph
plt.show()
```



[ ]:
```
# On a DataFrame, the plot() method is convenient to plot all of the columns␣
↪with labels
df4 = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,columns=['A', 'B',␣
↪'C', 'D'])
df4 = df4.cumsum()
df4.head()
```

[ ]:
```
                   A         B         C         D
2000-01-01 -1.750676 -0.631666 -0.286245  0.116738
2000-01-02 -0.945290 -2.954609 -0.315168  0.882456
2000-01-03 -1.763598 -2.834895  0.001486  0.118155
2000-01-04 -3.152844 -4.084474  1.848818  1.086287
2000-01-05 -2.210792 -4.873050  1.552365  1.971197
```

[ ]:
```
df4.plot()
plt.show()
```