Python Cheatsheet

Contents

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

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.

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

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.

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

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

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

In [ ]:
```
# 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)))
```

4. String manipulation

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

In [ ]:
```
# 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
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
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
```

5. Lists, tuples, and dictionaries

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

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.).

In [ ]:
```
# 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]
```

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

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

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

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

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

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

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.

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

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.

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

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.

In [ ]:
```
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"])
```

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.

In [ ]:
```
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")
```

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

In [ ]:
```
# 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 i

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

In [ ]:
```
# 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)
```

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

In [ ]:
```
# 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
```

In [ ]:
```
# 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()
```

In [ ]:
```
# 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()
```

In [ ]:
```
# 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()
```

In [ ]:
```
# 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.")
```

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`.

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

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

In [ ]:
```
# 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")
```

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

In [ ]:
```
# 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))
```

10. Working with datetime

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

In [ ]:
```
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"))
```

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

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

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

Inspecting your array

In [ ]:
```
# 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 ty
d = np.ones((3,5))
```

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

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

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

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

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

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

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

Basic math operations

In [ ]:
```
# 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 ty
d = np.ones((3,5))
```

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

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

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

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

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

Aggregate functions

In [ ]:
```
# 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 ty
d = np.ones((3,5))
```

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

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

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

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

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

Subsetting, slicing, and indexing

In [ ]:
```
# 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 ty
d = np.ones((3,5))
```

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

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

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

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

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

Array manipulation

In [ ]:
```
# 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 ty
d = np.ones((3,5))
```

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

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

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

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

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

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

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

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.

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

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

In [ ]:
```
# 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
```

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

Viewing data

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

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

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

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

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

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

Subsetting, slicing, and indexing

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

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

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

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

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

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

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

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

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

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

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

Missing data

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

In [ ]:
```
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'])
```

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

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

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

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

File handling

In [ ]:
```
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'])
```

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

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

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

In [ ]:
```
pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'], engine='openp
```

Plotting

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

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

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

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

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

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