|  |
| --- |
| 1. Expressions and Variables |
| Python evaluates expressions. The results can be stored in variables, which can be reused. Spaces around operators are optional! |
| Simple mathematical expressions |
| 1+1 # evaluates to: 2 |
| # Text after ‘#’ is a comment (ignored by Python) |
| 1+2\*3 # 7 |
| (1+2)\*3 # 9 |
| 5\*\*2 # 25 (power) |
| Division |
| 5/2 # 2.5 |
| 5//2 # 2 (integer division) |
| 5%2 # 1 (modulo, remainder) |
| Variables |
| x = 2 # Assignment: x is now 2 |
| hans = 1+x # hans is now 3 |
| z = x+hans # z is now 5 |
|  |
| 2. Strings and Printing |
| Strings are sequences of characters; a representation of text. Printing refers to the output of text from the program. |
| Hello world |
| print('Hello world!') # prints: Hello world! |
| print("Hello world!") # prints: Hello world! |
| # Note: Only straight single (') or double (") quotes can be used (not mixed)! Backticks and accents ( ´ or ` ) will not work! |
| Hello world with variables |
| variable = 'Hello world!' |
| print(variable) # prints: Hello world! |
| f-strings |
| first = 'Albert' |
| last = 'Einstein' |
| sentence = f'I like {first} {last}.' |
| print(sentence) # I like Albert Einstein. |
| print(f'pi: {3.14159:.2f}') # format pi: 3.14 |
|  |
| 3. User Input |
| Programs can prompt (ask) the user to enter a value. The value is always stored as a string (i.e., text) and you may need to convert it. |
| Prompting for a value |
| name = input("What's your name? ") |
| # Note: Careful use of " and ' allows the printing ' in a string! |
| print(f'Hello {name}') |
| Prompting for a numerical value |
| age = input("How old are you? ") |
| age = int(age) # convert to integer number |
| pi = input("What's the value of pi? ") |
| pi = float(pi) # convert to a decimal number |

|  |
| --- |
| 4. Conditional Tests (Comparisons) |
| Conditional tests evaluate to whether a statement is ***True*** or ***False***. |
| Conditional tests |
| 1 < 2 # True |
| 1+1 >= 2 # True |
| 1 < 2 < 3 # True (chaining) |
| 1 == 1 # True (1 equals 1) |
| # Note: Double equal signs (==) have to be used for equality testing! |
| 1 != 2 # True: 1 is not equal 2 |
| Boolean expressions |
| x = 7 |
| x < 10 or x > 15 # True |
| x < 10 and x > 7 # False |
| x < 10 and not x > 7 # True |
| x = 2 # Assignment: x is now 2 |
| hans = 1 + 1 # hans is now 2 |
| z = x + hans # z is now 4 |
|  |
| 5. Lists |
| Lists are a container for data. They contain multiple items in a particular order. |
| Creating a list |
| numbers = [1, 2, 3, 4, 5] |
| names = ['Alice', 'Bob', "Hans"] |
| empty\_list = [] |
| Get items from a list (indexing) |
| print(names[0]) # Alice |
| print(names[2]) # Hans |
| print(names[-1]) # Hans ([-1] -> last item) |
| names[0] = 'Al' # ['Al', 'Bob', 'Hans'] |
| Adding items to a list |
| numbers.append(42) # [1,2,3,4,5,42] |
| more\_names = ['Joe', 'Eve'] |
| names.extend(more\_names) |
| # ['Al', 'Bob', 'Hans', 'Joe', 'Eve'] |
| [1,2] + [3,4] # [1,2,3,4] |
| Slicing a list (getting a range from a list) |
| names[0:2] # ['Alice', 'Bob']  # (exclusive index 2; no "hans"!) |
| numbers[:] # [1,2,3,4,5,42] |
| numbers[:2] # [1,2] |
| numbers[:-3] # [1, 2, 3] (skip the last 3) |
| numbers[4:] # [5,42] |
| numbers[0:5:2] # [1,3,5] (step: 2) |
| numbers[::-1] # [42,5,4,3,2,1] (step: -1) |

|  |
| --- |
| List comprehensions (to create lists) |
| even\_numbers = [2\*x for x in range(10)] |
| odd\_numbers = [2\*x+1 for x in range(10)] |
| div\_by\_3 = [x for x in range(100) if x%3==0] |
| Conditional tests with lists |
| 2 in numbers # True |
| 17 in numbers # False |
| 'Charlie' not in names # True |
| Removing items from lists |
| numbers = [1, 2, 3, 4, 5, 42] |
| numbers.remove(42) # now: [1, 2, 3, 4, 5] |
| del numbers[3:5] # now: [1, 2, 3] |
| Copying lists |
| x = [1,2,3] |
| y = x # y refers to the same list as x |
| y[0] = 10 # modify y |
| print(x) # [10,2,3] – x was modified, too! |
| z = x[:] # z is a copy of x |
| z[0] = 5 # only z is modified, not x |
| Length of a list |
| x = [0,1,2,3,4] |
| len(x) # 5 |
|  |
| 6. Modules |
| Python comes with an extensive standard library of useful functions grouped into modules. Refer to the online documentations! |
| Importing a module |
| import math # now we can use math functions |
| print(math.exp(1)) # 2.718281828459045 |
| print(math.cos(0)) # 1.0 |
| Importing functions from a module |
| from math import exp as e |
| from math import cos, pi |
| print(e(0)) # 1.0 (no math. needed) |
| print(cos(pi)) # -1.0 |
|  |
| 7. If-statements |
| If-statements allow conditional execution of your code |
| Simple tests |
| if age >= 18: |
| print('You can vote!') |
| if-elif-else case distinctions |
| if age < 4: # do not forget the colon (:)! |
| ticket\_price = 0 |
| elif age < 18: # you can chain multiple elif |
| ticket\_price = 10 |
| else: |
| ticket\_price = 15 |

|  |
| --- |
| 8. Loops |
| Loops allow repeating certain lines of your code a certain number of times or while a condition is fulfilled. |
| For-loop |
| my\_list = [] |
| for number in range(3): |
| my\_list.append(number) |
| print(my\_list) # [0,1,2] |
| For-loop: continue |
| my\_list = [] |
| for number in range(5): |
| if number == 3: |
| continue # skips current for-iteration |
| my\_list.append(number) |
| print(my\_list) # [0,1,2,4] |
| Iterating over elements of a list |
| my\_dogs = ['Rex', 'Snoopy', 'Rufus'] |
| for dog in my\_dogs: |
| print(f'{dog} is the best dog ever!') |
|  |
| for number, dog in enumerate(my\_dogs): |
| print(f'{dog} is my dog no. {number}') |
| Progressbars  import time # provides time.sleep() |
| from tqdm import tqdm # creates progressbar |
| for dog in tqdm(my\_dogs): |
| print(f'{dog} is the best dog ever!')  time.sleep(1.5) # pause for 1.5sec |
| #67%|██████▋ | 2/3 [00:03<00:01, 1.51s/it] |
| # shows remaining time & time/iteration |
| While-loop |
| my\_list = [] |
| x = 1 |
| while x < 10: |
| x \*= 2 # same as: x = x \* 2 |
| my\_list.append(x) |
| print(f'my\_list: {my\_list} x: {x}') |
| # my\_list: [2, 4, 8, 16] x: 16 |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| 9. Functions |
| You can reuse code by defining functions (similar to print(…)) |
| Defining and invoking (calling) functions |
| def say\_hi(): |
| print('Hi!') |
| say\_hi() # prints: Hi! |
| say\_hi() # prints again: Hi! |
| Defining functions with parameters |
| def greet(name):  print(f'Hi {name}!') |
| greet('Alice') # prints: Hi Alice! |
| greet(name='Hans') # prints: Hi Hans! |
| # naming parameters is optional, but recommended! |
| Multiple parameters |
| def print\_sum(x,y): |
| print(x+y) |
| print\_sum(x=1, y=2) # prints: 3 |
| Default parameters |
| def print\_big\_sum(x,y,z=1): |
| print(x+y+z) |
| print\_big\_sum(x=1, y=2) # prints: 4 |
| print\_big\_sum(x=1, y=2, z=0) # prints: 3 |
| Return values |
| def subtract(x, y=1): |
| return(x - y) |
| a = subtract(x=3) # returns 2 |
| b = subtract(x=3, y=3) # returns 0 |
| print(f'a: {a}, b: {b}') # a: 2, b: 0 |
|  |
| 10. Error Handling |
| Some conditions (e.g. user input) might bring your program into a state, where it cannot continue as normal (a crash). |
| Catching errors |
| age = input("What's your age? ") |
| try: # Place below what could go wrong |
| age = int(age) |
| except: |
| print('You did not enter a number!') |
| else: |
| print(f"You're {age} years young!") |
|  |
| 11. Numpy |
| A module to perform numerical operations. |
| Basics |
| import numpy as np |
| x = np.arange(start=1, stop=2.5, step=0.5) |
| print(x) # [1., 1.5, 2. ] |
| print(np.exp(x)) # [2.71… 4.48… 7.38… ] |
| y = np.zeros\_like(x) # same shape and dtype as x |
| z = np.zeros(shape=(10,10), dtype=np.uint8) |
| converted = z.astype(np.uint8) # convert to uint8 |

|  |
| --- |
| Comparisons & Masking |
| x = np.array([1, 2, 3, 4]) |
| y = x > 2 # array([False, False, True, True]) |
| x[y] = 0 # array([1, 2, 0, 0]) |
| z = x[y] # array([0, 0]) |
|  |
|  |
| 12. Matplotlib/pyplot |
| Visualizing (plotting) data graphically. |
| Lineplots |
| import matplotlib.pyplot as plt |
| x = np.arange(start=0, stop=10, step=0.1) |
| y = np.exp(x) |
|  |
| plt.figure() |
| plt.plot(x, y) |
| plt.show() |
|  |
| 2D images |
| path\_to\_image\_file = 'data/dapi.tif' |
| image = plt.imread(path\_to\_image\_file) |
|  |
| print(f'image has the type: {type(image)}') |
| print(f'image has the shape: {image.shape} (rgb)') |
|  |
| plt.figure() |
| plt.imshow(image) |
| plt.show() |
|  |

|  |
| --- |
| Histogram |
| plt.figure() |
| plt.hist(image.flatten()) # convert 2D -> 1D |
| plt.show() |
|  |
| Customizing/labeling axes |
| plt.figure(figsize=(4,4)) # size in inches |
| plt.hist(image.flatten()) # convert 2D -> 1D |
| plt.xlabel('Intensity') # label on x-axis |
| plt.ylabel('Counts') # label on y-axis |
| plt.title('DAPI – Histogram') # title |
| plt.xlim([0, 150]) # x-axis range |
| plt.ylim([0, 1000000]) # y-axis range |
| plt.show() |
|  |
| Grid subplot |
| number\_rows = 1 |
| number\_cols = 3 |
|  |
| plt.figure(figsize=(7,3)) |
| plt.subplot(number\_rows, number\_cols, 1) |
| plt.hist(image.flatten()) # left plot |
|  |
| plt.subplot(number\_rows, number\_cols, 2) |
| plt.imshow(image) # middle plot |
|  |
| plt.subplot(133) # omitting commas also works |
| plt.plot(x,y) # right plot |
|  |
| plt.suptitle('Big Title') # title for all plots |
| plt.tight\_layout() # adjust padding of subplots |
| plt.show() |
|  |

|  |
| --- |
| Mosaic subplot |
| axes = plt.figure().subplot\_mosaic( |
| """ |
| aaa |
| aaa |
| b.c |
| """ |
| ) # letters are axes, colons are empty spaces |
|  |
| plt.sca(axes['a']) # set current axis to 'a' |
| plt.plot(x,y) |
|  |
| plt.sca(axes['b']) # set current axis to 'b' |
| plt.imshow(image) |
|  |
| plt.sca(axes['c']) # set current axis to 'c' |
| plt.hist(image.flatten()) |
|  |
| plt.tight\_layout() |
| plt.show() |
|  |
| Saving/exporting figures |
| plt.savefig('data/fig.png') # saves latest figure |
| plt.savefig('data/fig.pdf') # vector graphic |
| Additional cheatsheets |
| https://matplotlib.org/cheatsheets/ |
|  |
| 13. Pandas |
| Data science/analysis platform for tabular data and time series. |
| Simple example |
| import pandas as pd |
| df = pd.DataFrame(  {"area" : [1, 4],  "intensity" : [5.2, 8.1]},  index = ["cell\_1", "cell\_2"]) |
| print(df) |
| # prints: area intensity  # cell\_1 1 5.2  # cell\_2 4 8.1 |
| Accessing data |
| area = df['area'] |
| Appending to DataFrame |
| df = df.append(  pd.DataFrame({'area':5, 'intensity':4},  index=['cell\_3'])  ) |

|  |
| --- |
| Writing DataFrame to disk |
| df.to\_pickle('data/df.pkl') # recommended |
| df.to\_csv('data/df.csv') |
| df.to\_excel('data/df.xlsx') # needs 'openpyxl' |
| Reading DataFrame from disk |
| df2 = pd.read\_pickle('data/df.pkl') |
| df2 = pd.read\_csv('data/df.csv') |
| df2 = pd.read\_excel('data/df.xlsx') |
| Additional cheatsheets |
| https://pandas.pydata.org/Pandas\_Cheat\_Sheet.pdf |
|  |
| 14. scikit-image |
| Collection of algorithms and functions for image processing. |
| Reading images |
| from skimage import io |
| path\_to\_image\_file = 'data/dapi.tif' |
| image = io.imread(fname=path\_to\_image\_file) |
| Writing images |
| filename\_out = 'data/dapi\_out.tif' |
| io.imsave(fname=filename\_out, arr=image) |
| Filter Examples |
| from skimage.filters import median |
| filtered = median(image) # run median filter |
| from skimage.filters import try\_all\_threshold |
| try\_all\_threshold(image) |
|  |
|  |
| Measurements with regionprops  from skimage.measure import regionprops\_table  props = regionprops\_table(labels\_clear,# segments  image, # intensities  properties=['label',  'area'])  df = pd.DataFrame(props) # table with measurements |
| Further References |
| https://scikit-image.org/docs/stable/user\_guide.html |
| https://scikit-image.org/docs/stable/index.html |

|  |
| --- |
| 15. Paths and Globbing |
| Paths and path-handling are common sources of breaking code. Use dedicated modules from os.path to avoid these. |
| Joining paths |
| import os |
| path\_to\_directory = 'data' # adding '/' optional |
| path\_to\_subdirectory = 'more\_data' |
| path\_to\_file = 'dapi.tif' |
| joined\_path = os.path.join(path\_to\_directory,  path\_to\_subdirectory,  path\_to\_file) |
| # on windows: data\more\_data\dapi.tif |
| # on linux: data/more\_data/dapi.tif |
| Checking if file exists |
| os.path.exists(path\_to\_file) # returns True/False |
| Extracting information from path |
| os.path.dirname(joined\_path) # data/more\_data |
| os.path.basename(joined\_path) # dapi.tif |
| os.path.splitext(path\_to\_file) # ('dapi', '.tif') |
| Finding files with glob |
| from glob import glob |
| path\_to\_folder = os.path.join('data','glob') |
| glob(os.path.join(path\_to\_folder, '\*')) |
| # finds all files in data/glob, here: |
| #['data\\glob\\1.tif',  # 'data\\glob\\2.png',  # 'data\\glob\\40x\_1.tif',  # 'data\\glob\\1.png'] |
|  |
| glob(os.path.join(path\_to\_folder, '\*.tif')) |
| # finds all files that end with .tif |
| #['data\\glob\\1.tif',  # 'data\\glob\\40x\_1.tif'] |
|  |
| glob(os.path.join(path\_to\_folder, '1\*')) |
| # finds all files which filename starts with 1 |
| # ['data\\glob\\1.tif', 'data\\glob\\1.png'] |
| glob(os.path.join(path\_to\_folder, '\*1\*.tif')) |
| # filename has to have a 1 end end with .tif |
| # ['data\\glob\\1.tif', 'data\\glob\\40x\_1.tif'] |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |