# Python training - basic

## **1. Built-in types**

### **1.1 Numeric types**

|  |  |
| --- | --- |
| notations | integer - 2, int('2'), int(2.0) |
| binary - 0b11, bin(3) |
| hex – 0x05, hex(5) |
| float – 1.3, 1., .1, 5.0e-4, float(1) |
| complex – 3.14j, complex(‘1+2j’) |
| operations | +, -, \*, /, % |
| // - floor division |
| \*\* - power |
| abs(a) – absolute value |
| sqrt(a) – square root |
| math.ceil(a), math.floor(a) |
| <<, >>, &, |, ~, ^ - binary operations |

### **1.2 Strings**

|  |  |
| --- | --- |
| format | str(5) |
| "string1" |
| 'string2' |
| """  string3""" |
| ASCII | ord('a') – get ASCII value |
| chr(97) – get character by ASCII value |
| printing | print "a, b, c, values: %s %d %1.2f" % (a, 5, 4.444) |
| print("First ", 'x', 'y') |
| print("Second {0},{1},{2}".format(1, 2, 3)) |
| print("Third {0},{1},{2}".format(\*"abc")) |
| name = 'foo'  print(f'my name is {name}') # python 3.6+ |
| joining | a = "abc" + "d" + "ef" # a = "abcdef" |
| a = ":".join(["abc", "d"]) # a = "abc:d" |
| a = '%s %d' % ("Test", 5) # a = "Test 5" |
| slicing | string = "test string"  a = string[1:3] # 1 to (3-1)  a = string[:-1:] # 1 to (4-1) with step 1  a = string[2:] # 2 to (4-1)  a = string[::-1] # reversed |
| splitting | a = "1;2;3;".split(";") # a =['1', '2', '3'] |
| replacing | string = "string"  a = string.replace('t', 'the') #a = 'sbing' |
| stripping | a = " test ".strip() # a = "test"  a = "test a".strip('a') # a = "test " |

### **1.3 Boolean**

|  |  |
| --- | --- |
| values considered as false | False |
| None |
| empty container or string |
| numeric 0 |

### **1.4 Containers**

|  |  |
| --- | --- |
| general information | ordered collections: list, tuple |
| unordered collections: set, dictionary |
| mutable: list, set, dictionary |
| immutable: tuple |
| len(collection) – returns number of elements in the collection |
| list | [1, 2, 3], list() |
| list.append(element)  # append one element |
| list.extend([element1, element2])  # append multiple elements |
| list2 = list1 + [element1, element2]  # concatenate |
| list1 = [2, 3]  a, b = list1 # unpack elements |
| list.remove(element) |
| tuple | (1, 2), tuple() |
| tuple1 = (2, 3)  a, b = tuple1 # unpack elements |
| dictionary | {}, {key : value}, dict() |
| dict.items(), dict.keys(), dict.values() |
| dict[key] = value |
| dict.get(key, [default]) |
| dict.clear() |
| del(dict[key]) |
| set | set(), {1, 2, 3} |
| set.add(element) |
| set.remove(element) |
| set.clear() |
| a = {1, 2}  b = {1, 3}  intersection: a & b  union: a | b  difference: a – b  symmetric difference: a ^ b |

### **Files**

|  |  |
| --- | --- |
| opening | file = open(<path\_to\_file>, <opening\_mode>) |
| opening modes: r, w, a, r+, w+, a+ |
| operations | file.write(string) |
| file.seek(2) # go to second byte |
| file.close() |
| file.readline() # read one line only |
| file.read() # read the whole file |
| file.readlines() # all lines to a list |

## **2. Flow control**

### **2.1 Operators**

|  |  |
| --- | --- |
| comparison | ==, !=, > , <, <=, >= |
| logical | and, or, not |
| membership | in, not in |
| identity | is, is not |

### **2.2 Conditions**

|  |  |
| --- | --- |
| if statement | if condition1:  code block 1  elif condition 2:  code block2  else:  code block 3 |
| conditional expression | result = value if condition else other value  a = 10 if b > 0 else 1000 |

### **2.3 Loops**

|  |  |
| --- | --- |
| while | while condition:  code block1  if condition2: break  if condition3: continue  else:  code block2 |
| for | for target in object: # assign object items to target  code block1 # loop body  if condition2: break # exit, skip else  if condition3: continue # go to the top  else: # optional  code block |
| items = ['a', 'b']  for item in items:  print(item) |
| for c in 'word':  print(c) |
| d = {1: 'one', 2: 'two'}  for key, value in d.items():  print(key, value)  for key in d:  print(key)  for value in d.values():  print(value) |
| for i in range(0, 3) # (0, 1, 2)  print(i) |
|  | for x, y in zip('abc', [1, 2, 3]):  print(x, y) |

## **3. Functions**

|  |  |
| --- | --- |
| definition | def add(param1, param2):  return param1 + param2  y = add(2, 3)  print(y) # 5 |
| function assigned to a variable | def f():  pass  x = f  x() |
| function passed as an argument | def g(function\_param):  function\_param()  g(f) |
| list of functions | functions\_list = [g, g, g, g]  for func in functions\_list:  func(f) |

### **3.1 Default and positional arguments**

|  |
| --- |
| def print\_many\_times(what, times=2):  print(times \* what)  print\_many\_times("spam") # spamspam  print\_many\_times("spam", 3) # spamspamspam |
| WARNING – don't use mutable objects as default parameters unless you want to share it between function calls, you can default to None instead  def f(a, L=None):  if L is None:  L = []  L.append(a)  return L |

### **3.2 Keyword arguments**

|  |
| --- |
| print\_many\_times(a="spam", times=4) |
| print\_many\_times(a="spam", 5) # invalid  print\_many\_times('e', what='r') # invalid |

### **3.3Additional arguments**

|  |
| --- |
| def f(first, default='3', \*args, \*\*kwargs):  print('first: %s' % first)  print('default: %s' % default)  print('args:')  for arg in args:  print(arg)  print('kwargs:')  for k, v in kwargs.items():  print(k, v)  f('first', default='default', k1=1, k2=2, k3=[3,4,5], key='value') |

### **3.4 Unpacking arguments**

|  |
| --- |
| def one\_two\_three(one, two, three):  print(one, two, three)  l = ['2', '1', '3']  one\_two\_three(\*l) |

### **3.5 Lambda expressions**

|  |
| --- |
| add = lambda a, b: a + b  result = add(1,2)  print(result) |

### **3.6 Built-in functions**

|  |  |
| --- | --- |
| map | def double(a):  return a\*2  doubled = map(double, [1,2,3,4])  print(list(doubled)) |
| filter | natural = filter(lambda x: x > 0, [-2, -1, 0, 1, 2])  print(list(natural)) |
| all | print(all([1,1,'s', 0])) # False  print(all([1,1,'s', 'b'])) # True |
| any | print(any([1,0,'', None])) # True  print(any(0, None, "")) # False |
| enumerate | seasons = ['Spring', 'Summer', 'Fall', 'Winter']  print(list(enumerate(seasons)))  #[(0, 'Spring'), (1, 'Summer'),  # (2, ''Fall'), (3, 'Winter')] |
| min/max | print(min([3,0,1,-5)] # -5  print(max([3,0,1,-5)] # 3 |
| sum | print(sum([1, 2, 3])) # 6 |

### **3.7 List/dict/set comprehensions**

|  |  |
| --- | --- |
| list | a = [x\*\*2 for x in range(10)] |
| dict | integers = [3, 1, 4, 6, 10]  strings = ['1', '5', '10', '3']  nums = {i : s for i in integers for s in strings if i == int(s)} |
| set | unique\_sums = {x + y for x in range(4) for y in range(4)} |