**Introduction to Programming**

**Chapter overview:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter 2** | Operators | | +, -, \*, /, \*\*(exponential), % (remainder),  // (floor devision dropping remainder)  ==, !=, <, <=, >, >= |
| Variable assignment | | quant = 1  print(quant) |
| Scalar data types | | Int, float, bool, str, None  type(x)  isinstance(x, int) check specific type |
| Type casting | | Str(x), int(1.99)=1, float(1), bool(0)=True,  empty strings evaluate to False and all other strings evaluate to True.  print('abc ' \* 3) prints three times |
| Naming conflicts | |  |
| The # sign | | Comments |
| Tab-completion | |  |
| **Chapter 3** | If statements | | if -1 > x:  print(‘x is small’)  elif x == -1:  print(‘x is -1))  else:  print(‘x is big)) |
| for loops | | for character in 'Python':  print(character)  for n in range(1, 11):  print(n \*\* (1/2)) |
| while loops | | number\_to\_divide = 1000000000000  counter = 0  while number\_to\_divide >= 1:  counter += 1  number\_to\_divide /= 2  print(counter, number\_to\_divide) |
| Additional control flow syntax | | pass is the "do nothing" statement in Python  break interrupts a loop  continue advances a loop to its next iteration  else in loops -  to define commands to be executed at the end of the loop, |
| **Chapter 4** | Tuples | Foundations | tpl = (1, 4.2, '2', True)  good inf loops and return function values  number of entries cannot be changed  Tuples are fixed-length and immutable (entries can not be changed) |
| Slicing | 0 indexing  tpl[0]  tpl[-1] last element  tpl[0:2] start:end  nested: tpl[-1][:2]  print(tpl[::2]) frequency (every second element)  tpl[::-1] reverse order |
| Operators | tpl\_2 = tpl + (False, "gamma", .5)  tpl\_3 \* 3  1 in tpl – in/ not in checks whether an object belongs to the data structure |
| Unpacking | a, b, c, d, e, f = tpl  print(a)  a, b, \*\_ = tpl - \*\_ assigns the rest of the variables |
| Functions | Print(), type() |
| Methods | Tpl.count(1) – counts number of 1s in tpl  Tpl.index(4.2) – gives the index of entry 4.2 in tpl |
| Lists | Foundations | abc = [1, 2, "2", "cat", "whale"]  empty\_list = []  tpl2lst = list(tpl) – can transform tuples to lists  lst2tpl = tuple(abc)- can transform lists to tuples  lists are variable-length and their entries can be modified after definition |
| Slicing | Same as in tuples |
| Redefining | abc[-2] = 'dog' |
| Methods | Count()  Index()  Append() single entries  Extend() combining lists  Insert() arguments entry position and value shift the rest  Pop(2) deletes at specific index  Remove(‘hi’) deletes first entry that matches  Sort()  Reverse() |
| Dictionaries | Foundations | dct = {1:"this is one", "two":2, 3.0:True, "four":['a', 'b', 'c']}  print(dct)  empty\_dict = {}  list(dct) – creates list of keys (drops values) |
| Slicing | dct[key] returns value |
|  | NOT + or \*, but in (checks key) |
| Methods | .keys() (cast into list or tuple)  .values()  for k, v in dct.items(): to iterate over both  .get(‘two’, ‘default’) returns the value associated with a key, or a default value if that key does not exist in the dictionary  dct.update ({‘key’:’value’}) merges dicts  .pop(‘key’) deletes entry  Del dct[‘key’] deletes as well |
| Sets | | Unordered collections of unique elements  X = {'h', 'a', 'm'}  Set() to transform list or tuple into set  X.intersection(Y) gives only elements that are in both sets  X.union(Y) joins them so that all values are in once  X.difference(Y) returns the values that are only in one of them |
| Strings as sequences | | Tuple(‘abc’) 🡪 (‘a’,’b’,’c’), works for sets and list  Str.count(‘e’) counts number of times the letter  Str.index(‘e’)  My\_str[-1], my\_str[:4] |
| Comprehensions | | new\_list = [ i \*\* 2 for i in range(0, 11) ]  dict\_comp = { <key expression> : <value expression> for <key>, <value> in <collection> if <condition> }  set\_comp = { <expression on object> for <object> in <collection> if <condition> } |
| Copying data structures | | data\_copy = data[:]  data\_copy = data.copy() |
| **Chapter 5** | Functions | |  |
| Common built-in functions | | Print()  Type()  Isinstance()  Int(), str(),…  Len() returns number of elements in object  Sorted(), sort()  Reverse(), sorted(x, reverse=true)  all() checks if all variable are true  any() checks if atleast one value is true  min()  max()  round() .5 is rounded to closest even number  range()  help() , ?function  input() opens prompt to pass information  eval() takes single mandatory argument str  exec() create many variable names using string manipulation  for i, v in enumerate(range(100, 106)):  exec( 'var\_' + str(i) + '=' + str(v) ) |
| Generators and iterations | | **iterator** is **a type of object**that can be iterated upon  **special functions** that can be used to create iterators from iterables take the name of **generators.**  num\_list\_itor = iter(num\_list)  for item in num\_list\_itor:  print(item)  enumerate() takes an iterable as an argument and creates an iterator of 2-element tuples, where the first element of each tuple is the index of the element in the iterable, and the second is the element itself  zip() takes multiple iterables as arguments and creates an iterator of tuples containing all elements with the same index  gntr = ( x \*\* 2 for x in num\_list if x > 5 )  next()  \* returns all rest |
| User-defined functions | | def return\_message(message) :  return message  def simple\_f (\*args) :  return args 🡪 arbitrary positional arguments  def f\_2 (\*\*kwargs):  return kwargs 🡪 arbitrary key arguments  key arguments are pwr = 4, lst = [3, 4, 6]  Nested  def outer(num\_list):  def inner(num):  factor\_count = 0  while num / 2 % 1 == 0:  num /= 2  factor\_count += 1  return factor\_count    return [ inner(n) for n in num\_list ]  returning functions |
| Variable scope | | Variables that we define in the main body of a script are called **global**  Variables that are created inside a function in either of the above ways are called **local** variables: they are only available within the function itself.  To define or change a global variable within a function, it is necessary to use the global keyword followed by the variable name or names. |
| Lambda functions | | sum\_x\_y = lambda x, y : x + y |
| **Chapter 6** | Programming styles | | Procedural: one after the other  Functional: functions  Object oriented: classes |
| Objects and classes | | An **object** is a **data structure** bundling static information (think variables) and associated actions or behavior (think functions). |
| Own objects | | class MyClass:  pass  instance\_1 = MyClass()  class MyClass:  some\_attribute = 3  def some\_method(self, message):  return message  instance\_1.some\_attribute  instance\_1.some\_method(‘message’)  self  \_\_init\_\_() constructor  Global variables are capitalized VAR1 not meant to change  Private attributes \_  class Circle:    \_PI = 3.14159    def \_\_init\_\_(self, radius\_value, color\_value = None):  self.radius = radius\_value  self.color = color\_value    def set\_radius(self, radius\_value):  self.radius = radius\_value  return self.radius    def get\_radius(self):  return self.radius  def area(self):  return Circle.\_PI \* self.radius \*\* 2  class NewCircle(Circle):  pass  The new class will have all attributes and methods of the inhereted class (including the constructor, constants, etc.). |
| **Chapter 7** | Docstrings | | Add function description using quotation marks  "<function description / docstring>" |
| Errors | |  |
| Exception handling | | try: block of code to be tested.  except: block to code to handle a possible error. It will only be executed if the code in try raised an exception (i.e. it failed).  else: code to be executed if no errors were raised by the code in try.  finally: code to be executed at the end regardless of what happened in the previous steps. |
| Raising exceptions | | def factorial(n):    if not isinstance(n, (int, float)):  raise TypeError('You should input a number.')    if not isinstance(n, int) or n < 0:  raise Exception('You did not input a natural number.')    if n > 1:  return n \* factorial(n-1)  else:  return 1 |
| **Chapter 8** | Modules | | code library, .py  import mymodule |
| OS | |  |
| Sys | |  |
| Collections | |  |
| Math | | import math  math.exp() / math.e  math.log()/ math.log10()  math.pi  math.ceil()  math.floor() |
| Statistics | | import statistics  statistics.mean()  statistics.median()  statistics.mode()  statistics.stdev() |
| Random | | import random  random.random() (0,1) uniform  random.randrange(x,y) integer between two numbers  random.sample(sample,iterable) draws elements from the iterable **without replacement** to create a sample of the desired size.  random.seed() replicateble |
| datetime | | import datetime  datetime.datetime.now()  dt\_today = datetime.datetime(2021, 10, 7)  dt = datetime.datetime.strptime('210101 3:32PM', '%y%m%d %I:%M%p')  dt.strftime('%A %B %Y, %d - %I:%M %p')  dt.date()  dt(time)  dt.replace(year=2020)  [Python strftime reference cheatsheet](https://strftime.org/) |
| **Chapter 9** | Other modules | | import matplotlib.pyplot as plt  import numpy as np  import pandas as pd  import statsmodels as sm  to install use pip |
|  | Matplot lib | | fig = plt.figure()  axes = fig.add\_axes([0,0,1,1])  axes.plot(x, y)  axes.plot(x, z);  axes.set\_title('sales and production over time')  axes.set\_xlabel('time')  axes.set\_ylabel('values');  for two plots in one  fig, axes = plt.subplots(2, 1)  ax.legend(["curve1", "curve2", "curve3"])  ax.legend(loc=0) loc gives positon  color, alpha=transparency, linewidth, linestyle, marker,  ax.text(60, .25, 'production', color='b', fontsize=16)  fig.savefig('myplot.pdf')  fill\_between()  scatter()  step()  bar()  hist()  stacked bar, bottom=menMeans,  pie() |
| **Chapter 10** | Numpy | | import numpy as np  np.array()  .ndim -> dimensions  .shape -> number of elements in each dimension  .size -> how many individual entrie  .dtype type of entries  .astype() change type  np.arrange(start,stop,step=1)  np.linspace(start,stop,N)  np.diag()  np.zeros()  np.ones()  np.random.rand()  np.random.randn()  vector v[4]  Matrix M[0,0], M[1,:], M[:,1]  M[1:4, 1:4]  A[::2, ::2] frequency slicing  x[x > 5]  A.T transpose  Np.dot() dot product  from numpy.linalg import inv  Inv()  A.mean() / A.mean(axis=1)  A.sum(axis=0/1)  A.min()  A.max()  A.std()  A.var()  A.cumsum()  A.cumprod()  A.any()  A.all()  A.sort() |
| **Chapter 11** | Git Bash | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  | |  |

**import os**

**cwd = os.getcwd() # Get the current working directory (cwd)**

**cwd**

**f"{int(i):02}"**