2 Basic Constructions

2.1 Prescribed Exercises

Exercise 2.2 *Generate odd numbers.*

Write a program that generates all odd numbers from 1 to `n`. Set `n` in the beginning of the program and use a `while` loop to compute the numbers. (Make sure that if `n` is an even number, the largest generated odd number is `n` $- 1$.) Name of program file: `odd.py`.

Exercise 2.3 *Store odd numbers in a list.*

Modify the program from Exercise 2.2 to store the generated odd numbers in a list. Start with an empty list and use a `while` loop where you in each pass of the loop append a new element to the list. Finally, print the list elements to the screen. Name of program file: `odd_list1.py`.

Exercise 2.4 *Generate odd numbers by the range function.*

Solve Exercise 2.3 by calling the `range` function to generate a list of odd numbers. Name of program file: `odd_list2.py`.

Exercise 2.9 *Generate equally spaced coordinates.*

We want to generate x coordinates between 1 and 2 with spacing 0.01. The i-th coordinate, x_i, is then $1 + ih$ where $h = 0.01$ and i runs over integers $0, 1, \ldots, 100$. Compute the x_i values and store them in a list. Hint: Use a `for` loop, and append each new x_i value to a list, which is empty initially. Name of program file: `coor1.py`.

Exercise 2.10 *Use a list comprehension to solve Exercise 2.9.*

The problem is the same as in Exercise 2.9, but now we want the x_i values to be stored in a list using a list comprehension construct (see Chapter 2.1.6). Name of program file: `coor2.py`.

Exercise 2.23 *Write some simple functions.*

Write three functions (Name of program: `hw_func.py`):

1. `hw1`, which takes no arguments and returns the string ’`Hello, World!`’
2. `hw2`, which takes no arguments and returns nothing, but the string ’`Hello, World!`’ is printed in the terminal window
3. `hw3`, which takes two string arguments and prints these two arguments separated by a comma

Use the following main program to test the three functions:

```
print hw1()
hw2()
hw3('Hello ', 'World!')
```

2.2 Advanced Exercises

Exercise 2.12 *Compute a mathematical sum.*

The following code is supposed to compute the sum $s = \sum_{k=1}^{M} \frac{1}{k}$:

```
s = 0; k = 1; M = 100
while k < M:
  s += 1/k
print s
```

This program does not work correctly. What are the three errors? (If you try to run the program, nothing will happen on the screen. Type `Ctrl-C`, i.e., hold down the Control (`Ctrl`) key and then type the `c` key, to stop a program.) Write a correct program. Name of program file: `compute_sum_while.py`.

There are two basic ways to find errors in a program: (i) read the program carefully and think about the consequences of each statement, and (ii) print out intermediate results and compare with hand calculations. First, try method (i) and find as many errors as you can. Then, try method (ii) for $M = 3$ and compare the evolution of `s` with your own hand calculations.

Exercise 2.14 *Use a for loop in Exercise 2.12.*

Rewrite the corrected version of the program in Exercise 2.12 using a `for` loop over `k` values is used instead of a `while` loop. Name of program file: `compute_sum_for.py`.

Exercise 2.24 *Write the program in Exercise 2.12 as a function.*

Define a Python function `s(M)` that computes the sum s as defined in Exercise 2.12. Name of program: `compute_sum_func.py`.

Exercise 2.42 *Find the max/min elements in a list.*

Given a list `a`, the max function in Python's standard library computes the largest element in `a`: `max(a)`. Similarly, `min(a)` returns the smallest element in `a`. The purpose of this exercise is to write your own `max` and `min` function. Use the following technique: Initialize a variable `max_elem` by the first element in the list, then visit all the remaining elements (`a[1:]`), compare each element to `max_elem`, and if greater, make `max_elem` refer to that element. Use a similar technique to compute the minimum element. Collect the two pieces of code in functions. Name of program file: `maxmin_list.py`.