Lab 3

Functions, Conditions And Loops

Sup'Biotech 3

Python

Pierre Parutto

October 9, 2016

Preamble

Document Property

Authors	Pierre Parutto
Version	1.0
Number of pages	8

Contact

Contact the assistant team at: supbiotech-bioinfo-bt3@googlegroups.com

Copyright

Contents

1 Introduction

In this third lab, we will continue investigating `if` and `while` constructions using functions.

2 Warm-up

2.1 Scalar Product

As we have seen in lab2, the scalar product between two 2D vectors $X_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ and $X_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$ is defined as:

$$X_1 \cdot X_2 = x_1 \times x_2 + y_1 \times y_2$$

Write a function `dot_prod(x1: int, y1: int, x2: int, y2: int) -> int` that returns the dot product between X_1 and X_2.

Example

```
>>> dot_prod(1,2,3,4)
11
>>> dot_prod(4,3,3,4)
24
```

Correction:

```
def dot_prod(x1, y1, x2, y2):
    return x1 * x2 + y1 * y2
```

3 Sums

3.1 Sum Of Multiples Of 3

Write a function `sum_mult3(m: int, n: int) -> int` that returns the sum of all multiples of 3 starting at `m` (included) and ending a `n` (included).

Example

```
>>> sum_mult_3(2, 12)
30
>>> sum_mult_3(5, 20)
60
```

Correction:

```
def sum_mult_3(m, n):
    res = 0
    i = m
    while i <= n:
        if i % 3 == 0:
            res = res + i
        i = i + 1
    return res
```

3.2 Sum Of Multiples Of 3 - 2

Write a function `sum_mult3_2(m: int, n: int) -> int` that returns the sum of all multiples of 3 starting at `m` (included) and ending a `n` (included). This time, You will use a step of 3 in your loop.

Example

```
>>> sum_mult_3_2(2, 12)
30
>>> sum_mult_3_2(5, 20)
60
>>> sum_mult_3_2(0, 10)
18
```

Correction:

```
def sum_mult_3_2(m, n):
    res = 0
    if m % 3 == 0:
        i = m
    else:
        i = m + 3 - m % 3

    while i <= n:
        res = res + i
        i = i + 3
    return res
```

3.3 Sum With Steps

Write a function `sum_step(m: int, n: int, s: int) -> int` that returns the sum starting at `m` (included) to `n` (included) with a step of `s`.

Example

```
>>> sum_step(1, 9, 3)
12
```

```
>>> sum_step(0, 100, 15)
315
```

Correction:

```
def sum_step(m, n, s):
    wowo = 0
    i = m
    while i <= n:
        wowo = wowo + i
        i = i + s
    return wowo
```

3.4 Double Sum

Consider the following formula:

$$s_n = \sum_{i=1}^{n} \sum_{j=1}^{n-i} (i + j)$$

Write a function `sum_sum(n: int) -> int` that returns the double sum presented in the above formula.

Example

```
>>> sum_sum(9841)
317587964720
```

Correction:

```
def sum_sum(n):
    res = 0
    i = 1
    while i < n:
        j = 1
        while j < n-i:
            res = res + i + j
            j = j + 1
        i = i + 1
    return res
```

4 Sequences

4.1 A Sequence

Consider the following sequence:

$$u_n = \begin{cases} -u_{n-1} + \frac{u_{n-1}}{2} & \text{if } n \text{ is multiple of 3 and is even} \\ \frac{u_{n-1}}{2} & \text{if } n \text{ is multiple of 3} \\ 2 * u_{n-1} & \text{if } n \text{ is even} \\ 3 + u_{n-1} & \text{otherwise} \end{cases}$$

Write a function `my_seq(n: int, u0: float) -> float` that returns the value $u(n)$ as defined above.

Example

```
>>> my_seq(7854, -313.24345)
308.74345
```

Correction:

```
def my_seq(n, u0):
    yolo = u0
    i = 1
    while i <= n:
        if i % 3 == 0 and i % 2 == 0:
            yolo = -yolo + yolo / 2.0
        elif i % 3 == 0:
            yolo = yolo / 2.0
        elif i % 2 == 0:
            yolo = 2 * yolo
        else:
            yolo = 3 + yolo
        i = i + 1
    return yolo
```

5 Computing π

5.1 First Sequence

The following formula called Madhava-Gregory-Leibniz sequence converges toward π for $n \to \infty$:

$$u_n = 4 \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$$

Write a function `pi_seq1(n: int) -> float` that returns u_n as defined above.

Example

```
>>> pi_seq1(10)
3.232315809405594
>>> pi_seq1(100)
3.1514934010709914
>>> pi_seq1(1000)
3.1425916543395442
```

```
>>> pi_seq1(10000)
3.1416926435905346
```

Correction:

```
def pi_seq1(n):
    approx_pi = 0
    k = 0
    while k <= n:
        approx_pi = approx_pi + (-1)**k / (2*k+1.0)
        k = k + 1
    return 4 * approx_pi
```

5.2 Second Sequence

Another way of computing the value of π is by using the three following sequences:

$$\begin{aligned} A_{n+1} &= \frac{A_n+B_n}{2} \\ B_{n+1} &= \sqrt{A_n * B_n} \\ C_{n+1} &= C_n - 2^n(A_n - A_{n+1})^2 \end{aligned}$$

With the initial values:

$$\begin{aligned} A_0 &= 1 \\ B_0 &= \sqrt{\frac{1}{2}} \\ C_0 &= \frac{1}{4} \end{aligned}$$

The Brent-Salamin formula gives the value of π as a function of the three previous series for $n \to \infty$ as:

$$\pi = \lim_{n\to\infty} \frac{(A_n + B_n)^2}{4C_n}$$

Write a function `pi_seq2(n: int) -> float` that computes the value of π using the formula above.

Remark: To be able to use the square root function you need to add the following line **at the beginning of your file**:

```
from math import sqrt
```

Remark: You need to compute the three sequences at once in the same loop.

Example

```
>>> pi_seq2(1)
3.1405792505221686
>>> pi_seq2(10)
3.141592653589794
```

```
>>> pi_seq2(100)
3.141592653589794
>>> pi_seq2(1000)
3.141592653589794
```

Correction:

```
from math import sqrt

def pi_seq2(n):
    A_pred = 1
    B_pred = sqrt(1/2.0)
    C_pred = 1/4.0

    A = 0.0
    B = 0.0
    C = 0.0

    k = 1
    while k <= n:
        A = (A_pred + B_pred) / 2.0
        B = sqrt(A_pred * B_pred)
        C = C_pred - 2**(k-1) * ((A_pred - A))**2

        A_pred = A
        B_pred = B
        C_pred = C

        k = k + 1

    return (A + B)**2 / (4*C)
```

5.3 Comparison

Compare the values obtained by `pi_seq` and `pi_seq2` for the same values of `n`. Also compare it to the more precise value $\pi \approx 3.1415926536$ rounded at the tenth digit.