Seminar 3

ES12003

2024–25

1 Home exercises

Exercise 1.1

Make a list of the numbers from one to one million, and then use `min()` and `max()` to make sure your list actually starts at one and ends at one million. Also, use the `sum()` function to see how quickly Python can add a million numbers.

Solution 1.1

```python
numbers = list(range(1, 10**6 + 1))
print(f'min = {min(numbers)}')
print(f'max = {max(numbers):,.0f}')
print(f'sum = {sum(numbers):,.0f}')
```

Exercise 1.2

Write a script to create a dictionary where the keys are numbers between 1 and 12 (both included) and the values are the square of the keys. The output should look as follows:

```
The square of  1 is   1.
The square of  2 is   4.
The square of  3 is   9.
...
The square of 12 is 144.
```

Solution 1.2

```python
# create an empty dictionary to store the numbers and their squares
d = dict()
for x in range(1, 13): # iterate through numbers from 1 to 12
    # calculate the square of each number and store it
    # in the dictionary 'd' with the number as the key
    d[x] = x**2
    print(f'The square of {x:2d} is {d[x]:3d}.')
```

Exercise 1.3

Money deposited in a bank account earns interest, and this interest accumulates as the years pass. Write a program that asks the user to enter the initial principal. Then, given an interest rate of 4%, the program calculates

a) how much the account will be worth ten years from now
b) how many years it takes for the initial principal to double

Solution 1.3

```python
principal = float(input("Enter the initial principal: ") )
initial_principal = principal
double_the_principal = 2 * principal
interest_rate = 0.04
year = 0 # initial principal deposited in year 0
# part a
for i in range(1,11):
    principal *= (1 + interest_rate)
    # print(f'value in year {i:2d} is {principal:.2f}')
print (f'The value in 10 years is: {principal:.2f}')
# part b
principal = initial_principal
while principal < double_the_principal:
    year += 1
    principal *= (1 + interest_rate)
    # print(f'value in year {year:2d} is: {principal:.2f}')
print(f'It takes {year} years to double the initial principal.')
```

Exercise 1.4

The world's simplest recursive definition is probably the factorial function:

$$\begin{aligned} 1! &= 1 \\ (n+1)! &= (n+1) \times n! \end{aligned}$$

Write a program that

a) asks the user for an integer number, and then,
b) using a `while` loop, computes and returns its factorial.

Solution 1.4

```python
x = int(input("Enter a positive integer number: "))
i = 1 # set loop variable
factorial = 1 # initialise factorial to 1
while i <= x: # test loop variable
    factorial *= i # keep running product
    i += 1 # increment loop variable (shorthand notation)
print(f'{x}! = {factorial}')
```

Exercise 1.5

Print all prime numbers[1] within a given range, say 2 to 100.

Solution 1.5

```python
start = 2
stop = 100
print(f'The prime numbers between {start} and {stop} are:')
for num in range(start, stop+1):
    for i in range(2, num):
        if (num % i) == 0: # check for factors
            # print(f'{num:3g} is not a prime (it can be divided by
            ↪ {i} without remainder)')
            break # not a prime number so break inner loop
    else:
        print(f'{num:3g}')
```

[1] A prime number is a number that can only be divided by itself and 1 without remainders.

2 Class exercises

Exercise 2.1

Ordinal numbers indicate their position in a list, such as 1st or 2nd. Most ordinal numbers end in *th*, except 1, 2, and 3.

a) Store the numbers 1 through 9 in a list.
b) Loop through the list.
c) Use an `if-elif-else` statement inside the loop to print the proper ordinal ending for each number. Your output should read `1st 2nd 3rd 4th 5th 6th 7th 8th 9th`, and each result should be on a separate line.

Exercise 2.2

Write a program to count the number of distinct characters present in the word `pneumonoultramicroscopicsilicovolcanoconiosis` (it's the longest word in English).

Exercise 2.3

The Fibonacci numbers are the numbers in the following integer sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

In mathematical terms, the sequence F_n (of Fibonacci numbers) is defined by the recurrence relation:

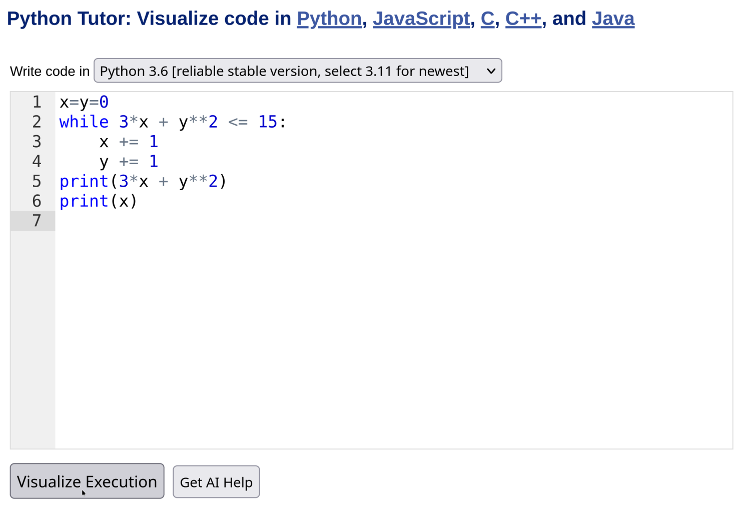
$$F_n = F_{n-1} + F_{n-2}$$

with initial values $F_0 = 0$ and $F_1 = 1$. Implement the sequence F_n using a `while` loop (e.g., print the first 20 numbers).

Exercise 2.4

The following code is from Topic 2, page 9.

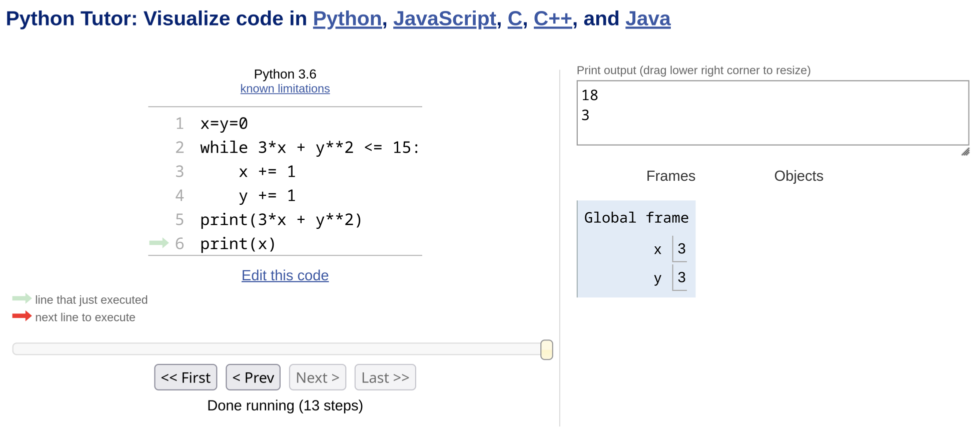
```
x=y=0
while 3*x + y**2 <= 15:
    x += 1
    y += 1
print(f'3*x + y**2 = {3*x + y**2}')
print(x)
```


Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java
Visualize Execution
Get AI Help

Write code in Python 3.6 [reliable stable version, select 3.11 for newest]

```
x=y=0
while 3*x + y**2 <= 15:
    x += 1
    y += 1
print(3*x + y**2)
print(x)
```

Figure 1: Using the Python tutor to visualise program execution.



```
x=y=0
while 3*x + y**2 <= 15:
    x += 1
    y += 1
print(3*x + y**2)
print(x)
```

Done running (13 steps)

Print output (drag lower right corner to resize)

```
18
3
```

Global frame

x	3
y	3

Figure 2: Entire program execution in Python tutor.

Visit the Python tutor (an online coding environment that allows you to visualise frame-by-frame how your code gets executed by the computer), select the Python language, enter the above code (or any code you find difficult to understand) in the box and click on `Visualize Execution` (see Figure 1).

Press `Next` to execute the first line. Press again `Next`. The code needs 13 steps to run (see Figure 2), where each step is one executed line of code. You can drag the slider to navigate forwards and backwards through all execution steps. On the right you can see the program printout and the final values of the variables `x` and `y` (execution is terminated as $3 \times 2 + 2^2 = 18 > 15$).

Finally, should you encounter bugs or simply want to tweak the code, click the `Edit this code` button below the code sample and restart the visualisation once you have made the changes you want.

Exercise 2.5

In home exercise 1.4, the factorial was implemented using a `while` loop. Write an equivalent program that uses instead a `for` loop.