Seminar 5

ES12003

2025–26

1 Home exercises

Exercise 1.1

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.

Solution 1.1

The following code exemplifies flow control. Having generated a sequence of integers from 1 to 9, `list()` converts that range object into a list. Thus line 1 sets up the data we want to iterate over.

```
1  numbers = list(range(1,10))
2  for number in numbers:
3      if number == 1:
4          print("1st")
5      elif number == 2:
6          print("2nd")
7      elif number == 3:
8          print("3rd")
9      else:
10         print(f"{number}th")
```

The `for` loop goes through each element in the `numbers` list. The code executed during each iteration of the `for` loop (lines 3 to 10) checks the value of `number` and prints its ordinal form. Note how different values of `number` are handled when printing ordinal numbers:

- `1st`, `2nd`, `3rd` are special cases/exceptions, because the ordinal values of 1, 2, and 3 are irregular in English.
- all other numbers follow a regular pattern: these numbers use the standard `th` suffix, so they can be handled with a single rule.

Note that in Python it is syntactically required to indicate which lines belong to which block (like inside a for loop or an if statement).

- All the `if`, `elif`, and `else` blocks are part of the `for` loop because they are indented under it.
- The `print()` statements are part of their respective conditional blocks because they are indented under them.

Finger exercise: Try to figure out how you would handle numbers beyond 20 (where `21st`, `22nd`, `23rd`, etc. come into play).

Exercise 1.2

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 1.2

Part a uses a `for` loop to apply compound interest each year. Note that the value of `principal` (as supplied by the user) will be overwritten as line 7 updates—once per year—the value of `principal` by multiplying it with `(1 + interest_rate)`. Thus we need (on lines 2 and 3) to preserve its original value (`initial_principal`) and set up a target value (`double_the_initial_principal`) for comparison later in the program (see line 13).

```
1 principal = float(input("Enter the initial principal: "))
2 initial_principal = principal # placeholder
3 double_the_initial_principal = 2 * principal
4 interest_rate = 0.04
```

```
 5 # part a
 6 for i in range(1,11):
 7     principal *= (1 + interest_rate)
 8     # print(f'value in year {i:2d} is {principal:.2f}')
 9 print (f'The value in 10 years is: {principal:.2f}')
```

Line 11 resets the `principal`. We then use a `while` loop to find out how many years it takes for the investment to double.[1] Line 14 increments the year counter until the principal reaches twice its original value.

```
10 # part b
11 principal = initial_principal # restore user's input
12 year = 0 # initialise variable
13 while principal < double_the_initial_principal:
14     year += 1
15     principal *= (1 + interest_rate)
16     # print(f'value in year {year:2d} is: {principal:.2f}')
17 print(f'It takes {year} years to double the initial principal.')
```

Note as well the following.

- The commented-out `print` statements inside both loops are like optional tools. Uncomment them to see what's happening inside each loop—it's a great way to understand how values change over time.
- The final `print` statements are not indented under the two loops (since these lines are aligned with the left margin, they are outside the loop blocks), so they run only once (i.e. after all the yearly calculations are complete).

Exercise 1.3

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.

[1] We use a `for` loop when we know in advance how many times you want to repeat something. We use a `while` loop when we don't know how many times the loop will run.

Solution 1.3

Line 1 prompts the user to enter a number: `input()` reads it as a string and `int()` converts it to an integer (which is stored in variable `x`, this is the number whose factorial we want to compute).

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

`i` is the counter used in the `while` loop (it starts at 1 because factorials are defined from 1 upwards) and `factorial` is the variable that will hold the running product. The computation on line 5 is the core of the factorial calculation. The loop runs as long as `i` is less than or equal to `x`, which ensures we multiply all integers from `1` to `x`.

Finger exercise: If you were to add a print statement (as in the previous exercise) to keep track of how values change inside the loop, where exactly would you place that statement?

Exercise 1.4

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).

Solution 1.4

In the following code, `n` sets the total number of Fibonacci numbers we want to generate. On line 2 we subtract 2 from it because the first two Fibonacci numbers (0 and 1) are printed before the loop starts (so the loop only needs to generate the remaining `n - 2` numbers). Note the following.

- On line 4, `end = " "` will keep the output on the same line (because space, instead of newline, is the string appended after the last value).
- `fn` and `sn` denote the first and second numbers in *any* pair of numbers in the sequence.

```
1 n = 12
2 n -= 2
3 fn, sn = 0, 1
4 print(fn, sn, end = " ")
5 count = 0
6 while count <= n:
7     print(fn + sn, end = " ")
8     fn, sn = sn, fn + sn
9     count += 1
```

The `count` variable counts how many additional Fibonacci numbers have been printed. When it becomes equal to (the updated value of) `n`, the loop will be terminated.

On line 8 we use tuples to update *pairs* of numbers: `fn` becomes the old `sn` and `sn` becomes the new number (that is `fn` plus `sn` in the previous pair).

Finger exercise: print `count` to obtain the value for which the conditional on line 6 became `False`.

Exercise 1.5

Have a look a the Sieve of Eratosthenes algorithm. Then write a program that prints all the primes smaller than or equal to a given number, n. For example, if $n = 10$, the output should be `2, 3, 5, 7`. If $n = 20$, the output should be `2, 3, 5, 7, 11, 13, 17, 19`.

Solution 1.5

```
num = 20

prime = [True for i in range(num+1)]

p = 2
while (p * p <= num):
```

```python
    if (prime[p] == True):
        for i in range(2 * p, num + 1, p):
            prime[i] = False
    p += 1

# print all prime numbers
for p in range(2, num+1):
    if prime[p]:
        print(p)
```

2 Class exercises

Exercise 2.1

The following code is from Topic 2, page 9.

```python
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)
```

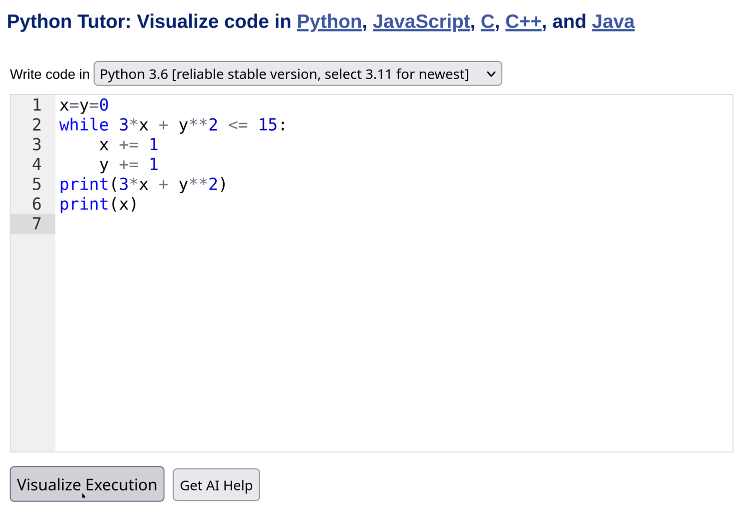
Visit the Python tutor (an online coding environment that allows you to visualise frame-by-frame how your code gets executed by the computer), select the Python language, enter the above code (or any code you find difficult to understand) in the box and click on `Visualize Execution` (see Figure 1).

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 2.2

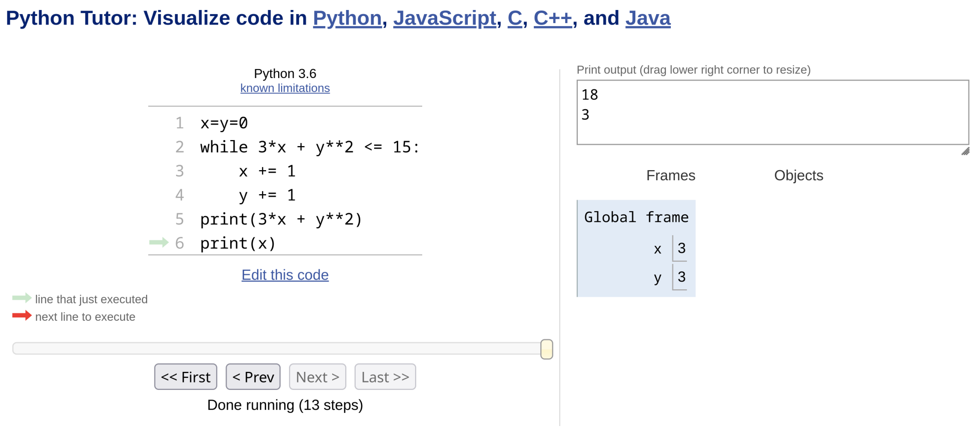
In home exercise 1.3, the factorial was implemented using a `while` loop. Write an equivalent program that uses instead a `for` loop.


Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

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.


Python Tutor: Visualize code in Python, JavaScript, C, C++, and Java

Python 3.6

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

Print output (drag lower right corner to resize)

```
18
3
```

Global frame

x	3
y	3

Done running (13 steps)

Figure 2: Entire program execution in Python tutor.

Solution 2.2

```python
x = int(input("Enter a positive integer number: "))
factorial = 1
for i in range(1, x+1, 1):
    factorial *= i
print(f'{x}! = {factorial}')
```

Exercise 2.3

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

Solution 2.3

```python
word = "pneumonoultramicroscopicsilicovolcanoconiosis"
new_letters = ""
for letter in word:
    if letter not in new_letters:
        new_letters += letter
print(len(new_letters))
```