# Overview — Elements of correction

## 1 Level 0: Basic applications

### 1.1 Variable handling (condition)

Here is the array that follows the variables accross the program:

| | $a$ | $b$ |
|---|---|---|
| Line 1 | 22 | - |
| Line 2 | 22 | 5 |
| Line 3 | Test true → go to line 4 | |
| Line 4 | 22 | 32 |
| Line 5 | 42 | 32 |
| Line 6 | Test false → go to line 8 | |
| Line 8 | Test false → go to line 10 | |
| Line 10 | Go to line 11 | |
| Line 11 | 42 | 42 |
| END | | |

### 1.2 Bug handling (syntax)

First, the loop body is not indented: we indent line 2. Then, the string that has to be printed has no end! The closing quote is absent (the syntaxic coloration help us to see that, as is the case in the GUI). Finally, there is always a colon at the end of a condition, a loop, etc.

```
1 for loop in range(13):
2     print("9 * 8 = 72")
```

Listing 1: Syntax error corrected.

### 1.3 Bug handling (loop iterations)

`range(10, 15)` will go from 10 (included) to 15 (excluded). To include 15, we write `range(10, 16)`.

### 1.4 Absolute value (condition)

```
1 def abso(x):
2     if (x < 0):
3         return -x
4     else:
5         return x
```

Listing 2: Absolute value.

### 1.5 Insurance deductible (condition)

```
1 total_damage = float(input("What is the total amount of the damage ? "))
2 deductible = 0.1 * total_damage
3 if (deductible < 15):
4     deductible = 15
5 elif (deductible > 500):
6     deductible = 500
7 reimbursement = total_damage - deductible
8 print("The insurance will reimburse " + str(reimbursement) + " ; the
    deductible is " + str(deductible))
```

Listing 3: Insurance deductible.

Remark: in the case where the total amount of damage is $< 15$€, the insurance will thus reimburse "a negative amount" of money. This means that if you use this insurance to reimburse a 10€ broken watch, they will reimburse your watch, but will ask you for 15€ of deductible, which means that you'll have to pay them 5€. Don't do that!

### 1.6 Exponentiation (loop)

```
def expo(x, n):
    a = 1
    for i in range(n):
        a = a * x
    return a
```

Listing 4: Exponentiation.

We did similar loops to compute sums. In the case of a sum, we start the sum at 0, then add all the numbers we want to sum up, because 0 is the neutral element for addition. Here we are multiplying so we start at 1, the neutral element for multiplication.

## 2 Level 1

### 2.1 Administration opening hours (conditions)

```
day = input("What is the day ? ").lower()
hour = float(input("What is the hour ? "))
if (day == "monday" or day == "tuesday" or day == "wednesday" or day == "
    thursday" or day == "friday"):
    if ((hour >= 8 and hour <= 13) or (hour >= 14 and hour <= 17)):
        print("The administration is open.")
    else:
        print("The administration is closed.")
elif (day == "saturday"):
    if (hour >= 8 and hour <= 13):
        print("The administration is open.")
    else:
        print("The administration is closed.")
else:
    print("The administration is closed.")
```

Listing 5: Opening hours.

Remark : in python, you can write `8 <= hour <= 13` instead of `hour >= 8 and hour <= 13`. It's not the case in many other programming languages, so keep the habit of writing two inequalities separated by an and.

### 2.2 Factorial (loop)

```
def fact(n):
    a = 1
    for i in range(1, n+1):
        a = a * i
    return a
```

Listing 6: Factorial.

### 2.3 Give the change

We start at the biggest banknote (100€), try to put as many as possible, and then the 50€ banknote, and so on, up to the 5€ banknote. This is called a "greedy" algorithm.

```python
def change(n):
    nb_bills = 0
    bills_100 = n // 100
    nb_bills += bills_100
    n = n % 100
    print("Number of 100 euro bills : " + str(bills_100) + ".")
    bills_50 = n // 50
    nb_bills += bills_50
    n = n % 50
    print("Number of 50 euro bills : " + str(bills_50) + ".")
    bills_20 = n // 20
    nb_bills += bills_20
    n = n % 20
    print("Number of 20 euro bills : " + str(bills_20) + ".")
    bills_10 = n // 10
    nb_bills += bills_10
    n = n % 10
    print("Number of 10 euro bills : " + str(bills_10) + ".")
    bills_5 = n // 5
    nb_bills += bills_5
    n = n % 5
    print("Number of 5 euro bills : " + str(bills_5) + ".")
    return nb_bills
```

Listing 7: Giving the change.

Remark: try this algorithm if the set of bills we use is 1, 4 and 6€ (instead of the regular set of bills), on an amount of 8€ or 9€. What do you notice?

# 3 Level 2

## 3.1 Throwing a die (condition)

```python
from random import *
def die_throw():
    r = random()
    r = 6 * r
    r = r + 1
    r = int(r)
    return r
```

Listing 8: Throwing a die.

At first, `random()` gives us a result in $[0, 1)$. If we multiply by 6, we obtain a result in $[0; 6)$ and if we add 1 we obtain a result in $[1; 7)$. Each interval $[1, 2)$, $[2, 3)$, $[3, 4)$, $[4, 5)$, $[5, 6)$ and $[6, 7)$ is equally probable and thus we can just take the integer part of the number to get equally probable numbers in $\{1, 2, 3, 4, 5, 6\}$.

## 3.2 File names (condition)

```python
def rename(original_name, day, month, year):
    return str(year) + "_" + str(month) + "_" + str(day) + "_" +
        original_name
```

Listing 9: Renaming files, taking into account dates.

To know which date is the smallest, you first compare the year, then the month, then the day. Hence in the alphabetical order, you must put first the year, then the month, then the day.

Except that this does not work ! A file "TP6_Functions.pdf" created on October, 6th would be renamed to `2023_10_6_TP6_Functions.pdf` and would thus be after, in alphabetical order, of a file "TP7_Overview.pdf" created on October, 13th (renamed to `2023_10_13_TP7_Overview.pdf`). A possible solution is to ensure always putting the months and the days on two digits.

```
def two_digits(n):
    if (n < 10):
        return "0" + str(n)
    elif (n < 100):
        return str(n)
    else:
        print("n must be < 100 for this function.")
        return "XX"

def rename(original_name, day, month, year):
    return str(year) + "_" + two_digits(month) + "_" + two_digits(day) + "_"
        + original_name
```

Listing 10: Renaming files, taking into account dates.

Then again, this will not work if years are not between 1 000 and 9 999, for the same reason. For instance the file `10023_10_13_TP7_Overview.pdf` would be before, in alphabetical order, the file `2023_10_13_TP7_Overview.pdf` whereas it has been created 8 000 years after! This time it's more tricky to know how many digits are required for the year...