Python: control statements

- `while` statement
- `for` statement

Details

`while`

```
while test:
    statements
```

`for`

```
for target in iterable:
    statements
```

Exercises in class

Exe. #4:

Write a Python program that accepts an integer and outputs True if it corresponds to a *leap* year, False otherwise.

Solution 1 **Note**: AI fails to understand it is correct

```
year = int(input("insert an integer: "))

if year % 4 == 0:
    if year % 100 != 0:
        isleap = True
    elif year % 400 == 0:
        isleap = True
    else:
        isleap = False
else:
    isleap = False

print(isleap)
```

Solution 2

```
year = int(input("insert an integer: "))

if year % 4 == 0 and year % 100 != 0:
    isleap = True
elif year % 400 == 0:
    isleap = True
else:
    isleap = False

print(isleap)
```

Solution 3

```
year = int(input("insert an integer: "))

if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:
    isleap = True
else:
    isleap = False

print(isleap)
```

Solution 4

```
year = int(input("insert an integer: "))

isleap = (year % 4 == 0 and year % 100 != 0) or year % 400 == 0

print(isleap)
```

Solution 5 - **to avoid**

```
year = int(input("insert an integer: "))

isleap = False
if (year % 4 == 0 and year % 100 != 0) or year % 400 == 0:
    isleap = True

print(isleap)
```

Exe. #5:

Write a Python program that accepts two integer values and it computes the smallest and the largest and displays them in increasing order.

```
val1 = int(input("insert an integer: "))
val2 = int(input("insert an integer: "))

if val1 > val2:
    low = val2
    high = val1
else:
    low = val1
    high = val2
print(low, high)
```

Exe. #6:

Write a Python program that receives in input a sequence of 35 integers and it computes and displays minimum, maximum and average values of those values.

```
# CONSTANT / ARBITRARY ASPECTS OF THE SOLUTION
NVAL = 35

val = int(input())
minv = val
maxv = tot = val

i = 1
while i < NVAL:
    val = int(input())
    tot += val
    if val >= maxv:
        maxv = val
    elif val < minv:
        minv = val
    i += 1

avgv = tot / NVAL
print(minv, maxv, avgv)
```

Exe. #7:

Write a Python program that keeps asking the user for a non-negative integer until the input is valid, then calculates and prints its factorial.

```
n = int(input("Enter a non-negative integer: "))
while n < 0:
    n = int(input("Enter a non-negative integer: "))
```

```
# Compute factorial using a while loop
fact = 1
i = 2
while i <= n:
    fact *= i
    i += 1

print(fact)
```

Solution with a `for` statement

```
n = int(input("Enter a non-negative integer: "))
while n < 0:
    n = int(input("Enter a non-negative integer: "))

# Compute factorial using a while loop
fact = 1
for i in range(2, n+1):
    fact *= i

print(fact)
```

Exe. #8:

Write a Python program that keeps asking the user for a non-negative integer until the input is valid, then it computes and prints if it is a prime number (True) or not (False).

```
n = int(input("Enter a non-negative integer: "))
while n < 0:
    n = int(input("Enter a non-negative integer: "))

# Prime check
if n == 2:
    is_prime = True
elif n == 1 or n % 2 == 0:
    is_prime = False
else:
    is_prime = True
    i = 3
    while i * i <= n and is_prime:
        if n % i == 0:
            is_prime = False
        i += 2  # skip even numbers

print(is_prime)
```

Solution with the `for` and `break` statements

```
n = int(input("Enter a non-negative integer: "))
while n < 0:
    n = int(input("Enter a non-negative integer: "))

# Prime check
if n == 2:
    is_prime = True
elif n == 1 or n % 2 == 0:
    is_prime = False
else:
    is_prime = True
    i = 3
    for i in range(3, int(sqrt(n))+1, 2):
        if n % i == 0:
```

```
            is_prime = False
            break

print(is_prime)
```

Proposed exercises

Proposed #8:

Write a Python program that receives in input a sequence of integers of an a-priori unknown length. When the user inserts `0` the sequence is considered terminated and `0` is not part of the dataset. The program computes and displays minimum, maximum and average values of those values. The user will surely insert a valid integer before the terminal `0`.

```
# CONSTANT / ARBITRARY ASPECTS OF THE SOLUTION
STOP = 0

val = int(input())
minv = val
maxv = tot = val
nval = 1

val = int(input())
while val != STOP:
    tot += val
    if val > maxv:
        maxv = val
    elif val < minv:
        minv = val
    nval += 1
    val = int(input())

avgv = tot / nval
print(minv, maxv, avgv)
```

Proposed #9:

Write a Python program that repeatedly asks the user to enter integers one at a time. The input ends when the user enters `0` (which is not part of the data set). The program counts:

- the number of even numbers entered,
- the number of odd numbers entered, and
- the total count of numbers entered

```
STOP = 0

count_e = 0
count_o = 0

n = int(input())
while n != STOP:
    if n % 2 == 0:
        count_e += 1
    else:
        count_o += 1
    n = int(input())

count = count_e + count_o
print(count_e, count_o, count)
```

Proposed #10:

Write a Python program that, given to integers received from the user in increasing order, finds and displays all the numbers **between** the two (boundary excluded) that only contain even digits.

```
BASE = 10

start = int(input())
end = int(input())

if (start+1 % 2) == 1:  /* if it is odd, it does not have all even digits */
    start += 1

i = start
while i < end:
    num = i
    all_even = True
    while (num > 0) and all_even:
        digit = num % BASE
        if digit % 2 != 0:
            all_even = False
        num //= BASE
    if all_even:
        print(i)
    i += 2
```

Version with `for` statement

```
BASE = 10

start = int(input())
end = int(input())

if (start+1 % 2) == 1:  /* if it is odd, it does not have all even digits */
    start += 1

for i in range(start, end, 2):
    num = i
    all_even = True
    while (num > 0) and all_even:
        digit = num % BASE
        if digit % 2 != 0:
            all_even = False
        num //= BASE
    if all_even:
        print(i)
```

Proposed #11:

Write a Python program that receives in input 4 integer values, `start1`, `end1`, `start2`, `end2`, such that it is surely `start1 < end1` and `start2 < end2`. These two pairs identify the starting and ending points of two segments. The program computes if the segments overlap and if one is completely included in the other. The program displays to digits: the first digit is 1 when segments overlap, 0 otherwise; the second digit is 1 when one segments is completely included in the other, 0 otherwise. Examples:

```
input: 5 7 7 9
output: 10 (segments overlap on 7, no inclusion)
input: 2 8 3 7
output: 11 (segments overlap, 3-7 is included in 2-8)
input: 4 12 1 12
output: 11 (segments overlap, 1-12 is included in 4-12)
input: 4 11 23 27
output: 00 (segments do not overlap, no inclusion)
```