While loop

`while` loop repeats the sequence of actions many times until some condition evaluates to `False`. The condition is given before the loop body and is checked before each execution of the loop body. Typically, the `while` loop is used when it is impossible to determine the exact number of loop iterations in advance.

The syntax of the `while` loop in the simplest case looks like this:

```
while some condition:
    a block of statements
```

Python firstly checks the condition. If it is False, then the loop is terminated and control is passed to the next statement after the `while` loop body. If the condition is True, then the loop body is executed, and then the condition is checked again. This continues while the condition is True. Once the condition becomes False, the loop terminates and control is passed to the next statement after the loop.

For example, the following program fragment prints the squares of all integers from 1 to 10. Here one can replace the "while" loop by the `for ... in range(...)` loop:

```
i = 1
while i <= 10:
    print(i ** 2)
    i += 1
```

In this example, the variable `i` inside the loop iterates from 1 to 10. Such a variable whose value changes with each new loop iteration is called a counter. Note that after executing this fragment the value of the variable `i` is defined and is equal to `11`, because when `i == 11` the condition `i <= 10` is False for the first time.

Here is another example use of the `while` loop to determine the number of digits of an integer `n`:

```
n = int(input())
length = 0
while n > 0:
    n //= 10  # this is equivalent to n = n // 10
    length += 1
print(length)  # 4
```

On each iteration we cut the last digit of the number using integer division by 10 (`n //= 10`). In the variable `length`we count how many times we did that.

In Python there is another, easier way to solve this problem: `length = len(str(i))`.

Loop control flow

2.1. else

One can write an `else:` statement after a loop body which is executed *once* after the end of the loop:

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Loop ended, i =', i)
```

At the first glance, this statement doesn't seem to have sense, because the `else:` statement body can just be put *after* the end of the loop. "else" statement after a loop only has sense when used in combination with the instruction `break`. If during the execution of the loop Python interpreter encounters `break`, it immediately stops the loop execution and exits out of it. In this case, the `else:` branch is not executed. So, `break` is used to abort the loop execution during the middle of any iteration.

Here is a Black Jack-like example: a program that reads numbers and sums it until the total gets greater or equal to 21. The input sequence ends with 0 for the program to be able to stop even if the total sum of all numbers is less than 21.

Let's see how it behaves on the different inputs.

Version 1. The loop is exited normally after checking the condition, so the "else" branch is executed.

```
total_sum = 0

a = int(input())

while a != 0:

    total_sum += a

    if total_sum >= 21:

        print('Total sum is', total_sum)

        break

    a = int(input())

else:

    print('Total sum is less than 21 and is equal to', total_sum, '.')
```

Version 2. The loop is aborted by `break`, so the "else" branch is skipped.

```
total_sum = 0

a = int(input())

while a != 0:

    total_sum += a

    if total_sum >= 21:

        print('Total sum is', total_sum)

        break

    a = int(input())

else:

    print('Total sum is less than 21 and is equal to', total_sum, '.')
```

"Else" branch can also be used with the "for" loop. Let's look at the example when a program reads 5 integers but stops right when the first negative integer is met.

Version 1. The loop is exited normally, so the "else" branch is executed.

```
for i in range(5):
    a = int(input())
    if a < 0:
        print('Met a negative number', a)
        break
else:
    print('No negative numbers met')
```

Version 2. The loop is aborted, so the "else" branch isn't executed.

```
for i in range(5):
    a = int(input())
    if a < 0:
        print('Met a negative number', a)
        break
else:
    print('No negative numbers met')
```

2.2. continue

Another instruction used to control the loop execution is `continue`. If Python interpreter meets `continue` somewhere in the middle of the loop iteration, it skips all the remaining instructions and proceeds to the next iteration.

```
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found a number", num)
```

If the `break` and `continue` are placed inside several nested loops, they affect only the execution of the innermost one. Let's look at rather silly example to demonstrate it:

```
for i in range(3):
    for j in range(5):
        if j > i:
            # breaks only the for on line 2
            break
        print(i, j)
```

The instructions `break` and `continue` are discouraged, if you can implement your idea without using them. Here is a typical example of a *bad usage* of the `break`: this code counts the number of digits in an integer.

```
n = int(input())
length = 0
while True:
    length += 1
    n //= 10
    if n == 0:
        break
print('Length is', length)
```

It's cleaner and easier-to-read to rewrite this loop with a meaningful loop condition:

```
n = int(input())
length = 0
while n != 0:
    length += 1
    n //= 10
print('Length is', length)
```

Multiple assignment

In Python it is possible for a single assignment statement to change the value of several variables. Let's see:

```
a, b = 0, 1
```

The effect demonstrated above code can be written as:

```
a = 0

b = 1
```

The difference between the two versions is that multiple assignment changes the values of two variables simultaneously.

Multiple assignment is useful when you need to exchange the values of two variables. In older programming languages without the support of multiple assignment this can be done using the auxiliary variable:

```
a = 1

b = 2

tmp = a

a = b

b = tmp

print(a, b)

# 2 1
```

In Python, the same swap can be written in one line:

```
a = 1

b = 2
```

```
a, b = b, a

print(a, b)

# 2 1
```

The left-hand side of "=" should have a comma-separated list of variable names. The right-hand side can be any expressions, separated by commas. The left-hand side and the right-hand side lists should be of equal length.

Problem «List of squares» (Easy)

Statement

For a given integer N, print all the squares of integer numbers where the square is less than or equal to N, in ascending order.

Your solution

```
# Read an integer:

n = int(input())

for i in range(1, n+1):

    if i**2 <= n:

        print(i**2)
```

Suggested solution

```
n = int(input())

i = 1

while i ** 2 <= n:

    print(i ** 2)

    i += 1
```

Problem «Least divisor» (Easy)

Statement

Given an integer not less than 2. Print its smallest integer divisor greater than 1.

Your solution

```
n = int(input())

i = 2

while n % i != 0:

   i += 1

print(i)
```

Suggested solution

```
n = int(input())

i = 2

while n % i != 0:

   i += 1

print(i)
```

Problem «The power of two» (Medium)

Statement

For a given integer **N**, find the greatest integer **x** where 2x2x is less than or equal to **N**. Print the exponent value and the result of the expression 2x2x.

Don't use the operation `**`.

```
n = int(input())

exp = 0

num = 1

while num <= n:
   num = 2**exp
   exp +=1
print(exp-2, 2**(exp-2))
```

Suggested solution

```
n = int(input())
two_in_power = 2
power = 1
while two_in_power <= n:
   two_in_power *= 2
   power += 1
print(power - 1, two_in_power // 2)
```

Problem «Morning jog» (Easy)

Statement

As a future athlete you just started your practice for an upcoming event. Given that on the first day you run *x* miles, and by the event you must be able to run *y* miles.

Calculate the number of days required for you to finally reach the required distance for the event, if you increases your distance each day by 10% from the previous day.

Print one integer representing the number of days to reach the required distance.

Your solution

```
x= int(input())
y= int(input())
day = 1
while x < y:
   x *= 1.10
   day +=1
```

```
print(day)
```

Suggested solution

```
x = int(input())

y = int(input())

i = 1

while x < y:

   x *= 1.1

   i += 1

print(i)
```

Problem «The length of the sequence» (Easy)

Statement

Given a sequence of non-negative integers, where each number is written in a separate line. Determine the length of the sequence, where the sequence ends when the integer is equal to 0. Print the length of the sequence (not counting the integer 0). The numbers following the number 0 should be omitted.

Your solution

```
sum = 0

i = int(input())

while i != 0:

   sum += 1

   i = int(input())

else:

   print(sum)
```

Suggested solution

```
len = 0
```

```
while int(input()) != 0:
    len += 1
print(len)
```

Problem «The sum of the sequence» (Medium)

Statement

Determine the sum of all elements in the sequence, ending with the number 0.

Your solution

```
# Read an integer:
n = int(input())
sum = 0
while n != 0:
    sum += n
    n = int(input())
print(sum)
```

Suggested solution

```
sum = 0
element = int(input())
while element != 0:
    sum += element
    element = int(input())
print(sum)
```

Problem «The average of the sequence» (Medium)

Statement

Determine the average of all elements of the sequence ending with the number 0.

Your solution

```
# Read an integer:

n = int(input())

sum = 0

total = 0

while n != 0:

    sum   += n

    total += 1

    n = int(input())

print(sum/total)
```

Suggested solution

```
sum = 0

len = 0

element = int(input())

while element != 0:

    sum += element

    len += 1

    element = int(input())

print(sum / len)
```

Problem «The maximum of the sequence» (Medium)

Statement

A sequence consists of integer numbers and ends with the number 0. Determine the largest element of the sequence.

Your solution

```
# Read an integer:

n1 = int(input())

n2 = int(input())

while n1 > 0 and n2 > 0:

    if n1 > n2:

        n2 = int(input())

        result = n1

    else:

        n1 = int(input())

        result = n2

print(result)
```

Suggested solution

```
max = 0

element = -1

while element != 0:

    element = int(input())

    if element > max:

        max = element

print(max)
```

Problem «The index of the maximum of a sequence» (Hard)

Statement

A sequence consists of integer numbers and ends with the number 0. Determine the index of the largest element of the sequence. If the highest element is not unique, print the index of the first of them.

Your solution

```
n = int(input())

max_num = n

i = 1

index =1

while n != 0:

   n = int(input())

   i +=1

   if n > max_num:

      max_num = n

      index = i

print(index)
```

Suggested solution

```
max = 0

index_of_max = -1

element = -1
```

```
len = 1
while element != 0:
    element = int(input())
    if element > max:
        max = element
        index_of_max = len
    len += 1
print(index_of_max)
```

Problem «The number of even elements of the sequence» (Medium)

Statement

Determine the number of even elements in the sequence ending with the number 0.

Your solution

```
n = int(input())
count = 0
while n != 0:
    if n % 2 == 0:
        count +=1
    n = int(input())
print(count)
```

Suggested solution

```
num_even = -1
element = -1
while element != 0:
    element = int(input())
    if element % 2 == 0:
        num_even += 1
print(num_even)
```

Problem «The number of elements that are greater than the previous one» (Hard)

Statement

A sequence consists of integer numbers and ends with the number 0. Determine how many elements of this sequence are greater than their neighbours above.

Your solution

```
# Read an integer:
a = int(input())
b = int(input())
total = 0
while a > 0 and b > 0:
    if a < b:
        total += 1
    a = b
    b = int(input())
print(total)
```

Suggested solution

```
prev = int(input())
```

```
answer = 0
while prev != 0:
    next = int(input())
    if next != 0 and prev < next:
        answer += 1
    prev = next
print(answer)
```

Problem «The second maximum» (Hard)

Statement

The sequence consists of distinct positive integer numbers and ends with the number 0. Determine the value of the second largest element in this sequence. It is guaranteed that the sequence has at least two elements.

Your solution

```
n = int(input())
max_num = n
sec_max = n
i = 1
while n != 0:
    n = int(input())
    i+=1
    if n > max_num:
        sec_max = max_num
        max_num = n
    elif n > sec_max :
        sec_max = n
    elif i == 2:
        sec_max = n
print(sec_max)
```

Suggested solution

```
first_max = int(input())
second_max = int(input())
if first_max < second_max:
    first_max, second_max = second_max, first_max
element = int(input())
while element != 0:
    if element > first_max:
        second_max, first_max = first_max, element
    elif element > second_max:
        second_max = element
    element = int(input())
print(second_max)
```

Problem «The number of elements equal to the maximum» (Hard)

Statement

A sequence consists of integer numbers and ends with the number 0. Determine how many elements of this sequence are equal to its largest element.

Your solution

```
maximum = 0

num_maximal = 0

element = -1

while element != 0:

    element = int(input())

    if element > maximum:

        maximum = element

        num_maximal = 1

    elif element == maximum:

        num_maximal += 1

print(num_maximal)
```

Suggested solution

```
maximum = 0

num_maximal = 0

element = -1

while element != 0:

    element = int(input())

    if element > maximum:

        maximum, num_maximal = element, 1

    elif element == maximum:

        num_maximal += 1

print(num_maximal)
```

Problem «Fibonacci numbers» (Hard)

Statement

The Fibonacci sequence is defined as follows:

$$\phi_0=0,\ \phi_1=1,\ \phi_n=\phi_{n-1}+\phi_{n-2}.$$ϕ0=0, ϕ1=1, ϕn=ϕn−1+ϕn−2.

Given a non-negative integer nn, print the $n$nth Fibonacci number ϕ_nϕn.

This problem can also be solved with a `for` loop.

Your solution

```
a = 0

b = 1

n= int(input())

if n == 0:

   print(a)

elif n == 1:

   print(b)

else:

   for i in range(n-1):

      c = a + b

      a = b

      b = c

   print(c)
```

Suggested solution

```
n = int(input())

if n == 0:

   print(0)

else:

   a, b = 0, 1

   for i in range(2, n + 1):

      a, b = b, a + b

   print(b)
```

Problem «The index of a Fibonacci number» (Hard)

Statement

The Fibonacci sequence is defined as follows:

$$\phi_0=0,\ \phi_1=1,\ \phi_n=\phi_{n-1}+\phi_{n-2}.\phi0=0,\ \phi1=1,\ \phi n=\phi n-1+\phi n-2.$$

Given an integer aa, determine its index among the Fibonacci numbers, that is, print the number nn such that $\phi_n=a\phi n=a$. If aa is not a Fibonacci number, print -1.

Your solution

```
result = -1

a = 0

b = 1

c = a + b
```

```
n= int(input())

while n != c:

    for i in range(50):

        c = a + b

        a = b

        b = c

        if n == c:

            result = i+2

            break

        if i == 49:

            n = c

            break

print(result)
```

Suggested solution

```
a = int(input())

if a == 0:

    print(0)

else:

    fib_prev, fib_next = 0, 1

    n = 1

    while fib_next <= a:

        if fib_next == a:

            print(n)

            break

        fib_prev, fib_next = fib_next, fib_prev + fib_next
        n += 1

    else:

        print(-1)
```

Problem «The maximum number of consecutive equal elements» (Hard)

Statement

Given a sequence of integer numbers ending with the number 0. Determine the length of the widest fragment where all the elements are equal to each other.

Your solution

```
count = 0

max_len = 0

prev = -1

next = -1

while prev != 0:

    prev = int(input())

    if prev > next:

        count = 1

        next = prev

    elif prev == next:

        count += 1



    elif prev < next:
        count = 1
        next = prev
    if count > max_len:
        max_len = count
print(max_len)
```

Suggested solution

```
prev = -1
curr_rep_len = 0
max_rep_len = 0
element = int(input())
while element != 0:
    if prev == element:
        curr_rep_len += 1
    else:
        prev = element
        max_rep_len = max(max_rep_len, curr_rep_len)
        curr_rep_len = 1
    element = int(input())
max_rep_len = max(max_rep_len, curr_rep_len)
print(max_rep_len)
```