STEP IT ACADEMY

PROGRAMMING WITH python

Lesson # 6

The while Loop

CONTENTS

The while Loop

When creating programs, you don't always know how many times you'll have to repeat a set of actions. For this, there is a special type of loop: **conditional loops**.

Imagine that a character of a computer game has stepped on lava, and every second 5 health points are taken away. It will last until the player moves to another location.

The developer does not know how many iterations there should be in this case, so they use `while` to create a loop (Figure 1).

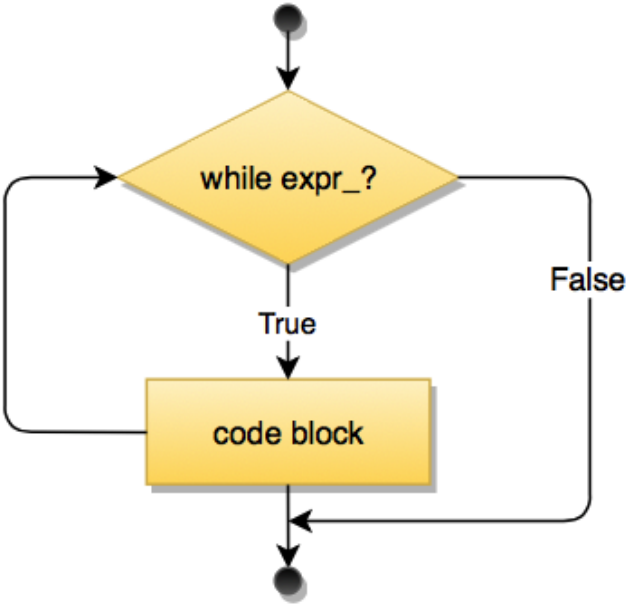

Figure 1

A `while` loop is a conditional loop. It differs from `for` in that it doesn't know how many times it will be executed. However, the `while` principle is very similar to `for`. First, the condition is checked; if it is true, the code is executed in the body of the loop, then the condition is checked again. If at some point it becomes false, the program exits the loop.

General syntax of `while` looks like this:

```
while conditional_statement:
   instructions
```

Let's look at an example of using a loop in the code:

```
   # example of the while loop
a = 5
while a<10:
  print("Now the value of a: " + str(a)
        + ". But it is less than 10!")
  a = a + 1 # Increasing the value of a by 1 in
              each iteration
print("That's it! Now the number a is greater
       than 10!")
```

This code constantly outputs value of the number that is stored in the `a` variable, then it increments it. This will continue until `a` is greater than 10.

You can use any conditions, both simple and complex.

Infinite Loop

You may need to perform certain actions continuously, for example, to juggle a ball. A **loop without set condition** is called **infinite**.

Consider an **example of an infinite loop**:

```
while True:
   instructions
```

You need to be very careful when using infinite loops, use them only if this is really unavoidable! If not properly applied, this loop can simply "freeze" the execution of your program. And in most cases you can do without such a loop if you create a suitable condition.

The break Statement

The `break` statement interrupts execution of the loop, for example, if the code uses an infinite loop as the one we saw earlier. This can be used in cases where program execution must be interrupted as soon as the condition is met.

Let's say you decided to beat the record of the world-famous football player Diego Maradona and to juggle a ball with your head over 7000 times. Thus, you start the `while` loop, and the current score is written to a variable. As soon as the ball touches the floor, the loop ends and no more values are added to the variable (see Figure 2 on page 6).

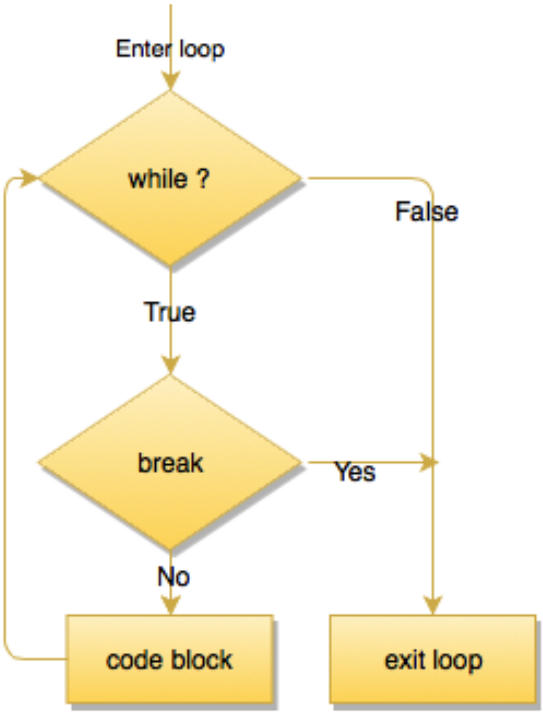

Figure 2

This is how the `break` is executed in the code (Figure 3):

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

Figure 3

The continue Statement

The `continue` statement ignores the code that follows it and continues the execution of the loop from the next iteration as if there is no code after `continue` (Figure 4).

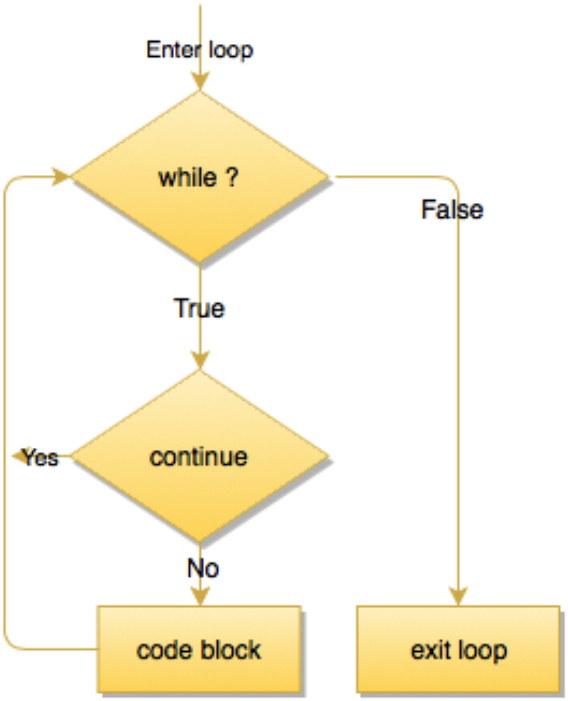

Figure 4

Figure 5 shows how it looks like in the code (see page 8). As we can see, the `continue` principle is significantly different from `break`.

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

Figure 5

Game: Guess My Number

So, you learned how to work with loops, and now it's time to consolidate your knowledge!

Let's create the game **Guess My Number**. The essence of the game is simple enough: the user must guess the number that the computer has come up with (Figure 6).

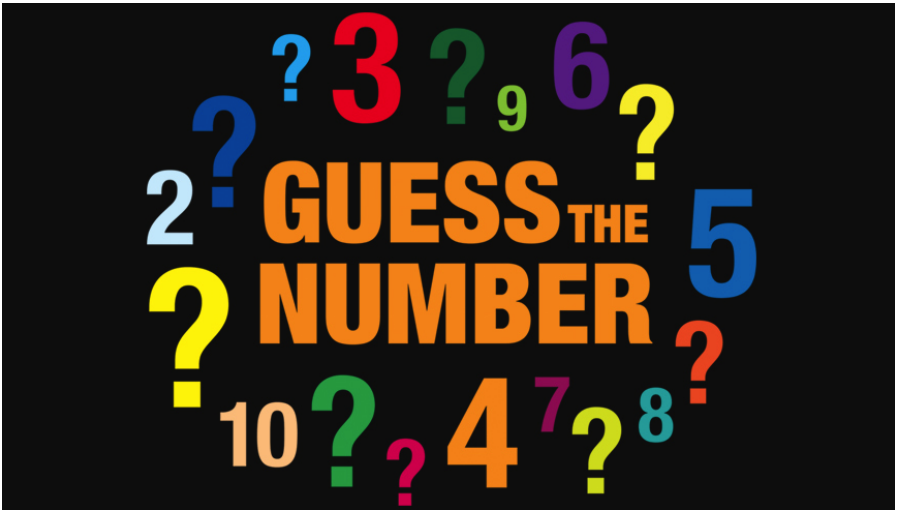

Figure 6

Listing 1

```
import random

print("-------Guess my number-------")
print("You need to guess the number picked by
       enigmatic computer!"
      "Number is in range from 1 to 10")
magic_number = random.randint(1, 20)
user_number = 0

while user_number != magic_number:
   user_number = int(input("Your number: "))
   if magic_number > user_number:
       print("The magic number is greater than
              yours!")
   elif magic_number < user_number:
       print("The magic number is less than
              yours!")
print(f"You guessed it right!
        Magic number: {magic_number}")
```

Let's see how our code works.

We added the `random` module to generate a random number. As you remember, the library should be added at the very beginning of the code.

We also created the `magic_number` (*number picked by the computer*) and `user_number` (*number entered by the user*) variables.

The basis of our game is a loop:

```
while user_number != magic_number:
```

This loop starts the game. It will constantly ask the user for the number it picked, until they guess it:

```
while user_number != magic_number:
  user_number = int(input("Your number: "))
print(f"You guessed You guessed it right!
        Magic number: {magic_number}")
```

We created a simplified version of **Guess My Number**. **Of course, you can leave it as it is, but let's improve the game!** Add information about how close the user is to the right number. This can be done by adding the `if-elif` condition to the loop.

Now we can not only check whether the user has guessed the number, but also indicate whether this number is greater or less:

```
while user_number != magic_number:
  user_number = int(input("Your number: "))
  if magic_number < user_number:
     print("The magic number is greater than
           yours!")
  elif magic_number > user_number:
     print("The magic number is less than yours!")
print(f"You guessed it right!
        Magic number: {magic_number}")
```

Lesson # 6
The while Loop