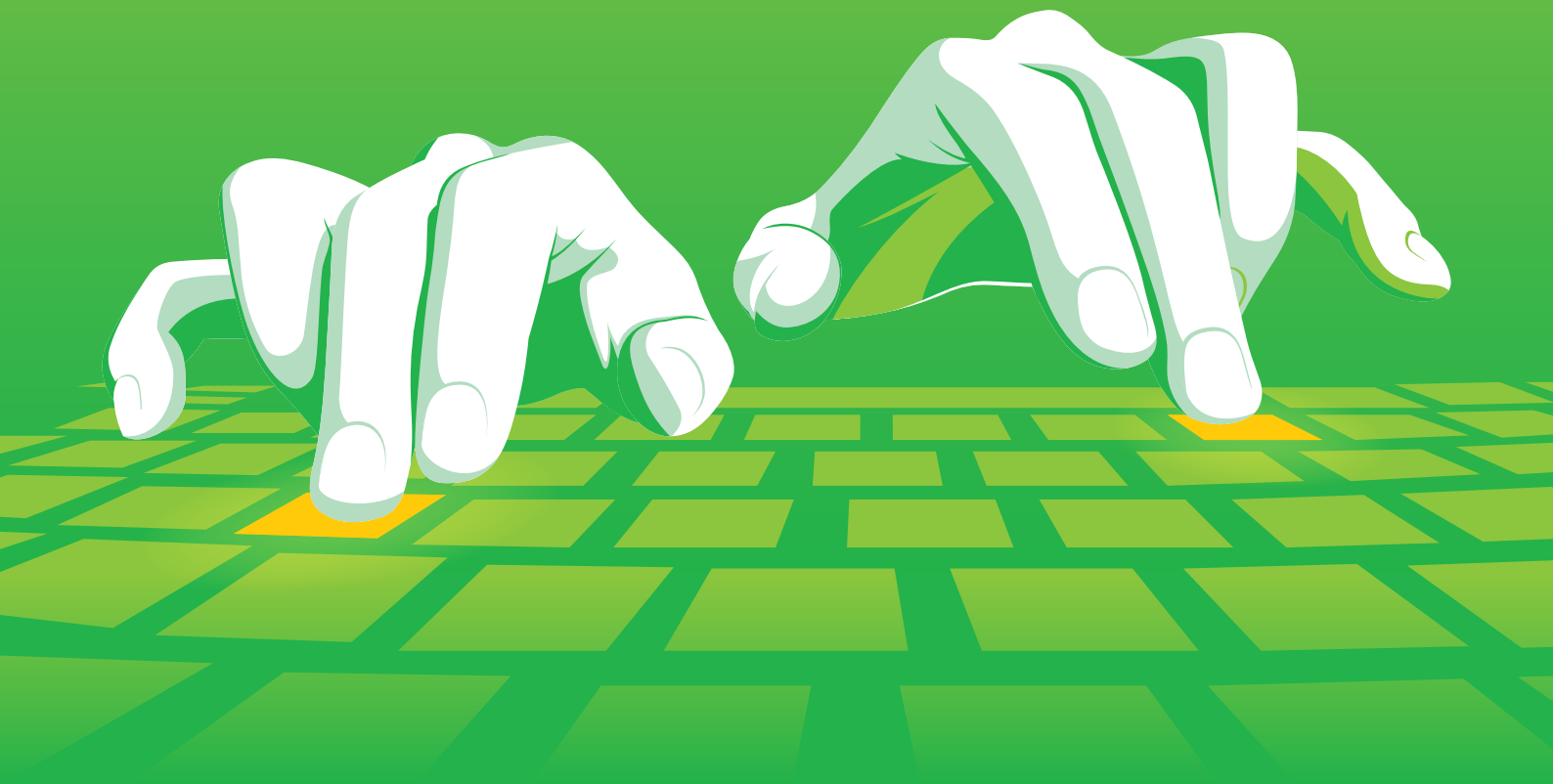
DIGITAL Teens

can_code

with Python 2



binarylogic

Lesson 3
Repetition

Sometimes in programming we need a part of the code to be repeated several times. Almost all the programming languages provide a function called a loop. Loops allow you to execute a single line of code or set of code statements several times. The number of repetitions could be specified as a certain number, or the repetitions could depend on a certain condition.

Loops

Sometimes in coding you need to repeat a set of commands multiple times. This takes a lot of time and effort. Python provides different control structures to help you avoid rewriting code commands.

Python supports two types of loops: the **for** loop and the **while** loop.

Loops in Python

for loop

```
for loop_varaible in range:
    statements
```

while loop

```
while condition:
      statements
```

for loop

The **for** loop is used when you want to repeat a set of commands a specified number of times. The number of repetitions is specified in `range()` parameters.

```
for loop_varaible in range():
    statements
```

The statements which are repeated have to be indented.

This specifies the number of repetitions.

Range() function

You use the `range( )` function with the **for** loop to specify the number of repetitions. The variable that counts the repetitions is called the counter.

In a `range( )` function the counter starts counting from **0**, increments by **1** and ends at the specified number. For example, in `range(5)`

```
# Prints out the value of i
for i in range(5):
   print(i)
```

```
0
1
2
3
4
```

You can specify the starting value by adding a parameter. For example, `range(2,5)`, means that the counter starts counting from **2** and ends at **4** (the value 5 is not included).

```
# Prints out the value of i
for i in range(2,5):
   print(i)
```

```
2
3
4
```

The default value of increment in `range( )` is 1, but you can specify the increment value by adding a third parameter in the range function. For example, `range(1,5,2)`, means the counter starts counting from **1**, ends at **5** and it increases by **2**.

```
# Prints out the value of i
for i in range(1,5,2):
   print(i)
```

```
1
3
```

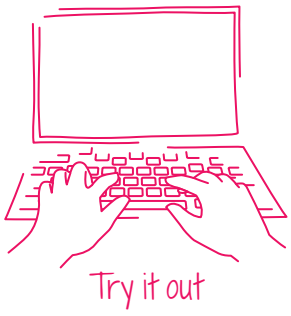
The third parameter in range is called the step.

ne value
(1,5,**2**):

The step can also be a negative number. In this case the count is reversed.

```
# Prints out the value of i
for i in range(10,5,-2):
   print(i)
```

```
10
8
6
```



Try out the following code and write down the values that appear on the screen.

```
for i in range(4,0,-1):
    print (i)
```

```
for i in range(0,10,2):
    print (i)
```

Students letter grades

In the previous lesson, we worked through an example calculating a students letter grades. The program checked if the student passed the exams. Let's see how we can apply the **for** loop. In the previous example there was one student. Let's assume now that you have to check the grades of a whole class of 15 students. The new parts of the code that you have to add are the following.

Syntax inspector
Be careful with the indentation!

```
for st in range (0,15):
    print("Please enter a student's name: ")
    name=input()
    print("Please enter a grade: ")
    g=int(input())
    if g>50:
        print(name,"passed the exams.")
        if g<=70:
            print(name,"got a C.")
        elif g<=90:
            print(name,"got a B.")
        else:
            print(name,"got an A.")
    else:
        print(name,"didn't pass the exams.")
```

To read each student's name.

The **for** loop is used when you know the number of the repetitions from the beginning. What happens when this number is not known and the repetition depends on a condition? For such cases, Python offers the **while** loop.

while loop

The **while** loop is used when the number of repetitions is not known in advance. As long as the condition is **True** the loop iterates. After every repetition the condition is checked. When the condition becomes **False**, the iteration stops and program control passes to the line following the loop. If the condition is initially **False**, the statements of the loop will not be executed at all.

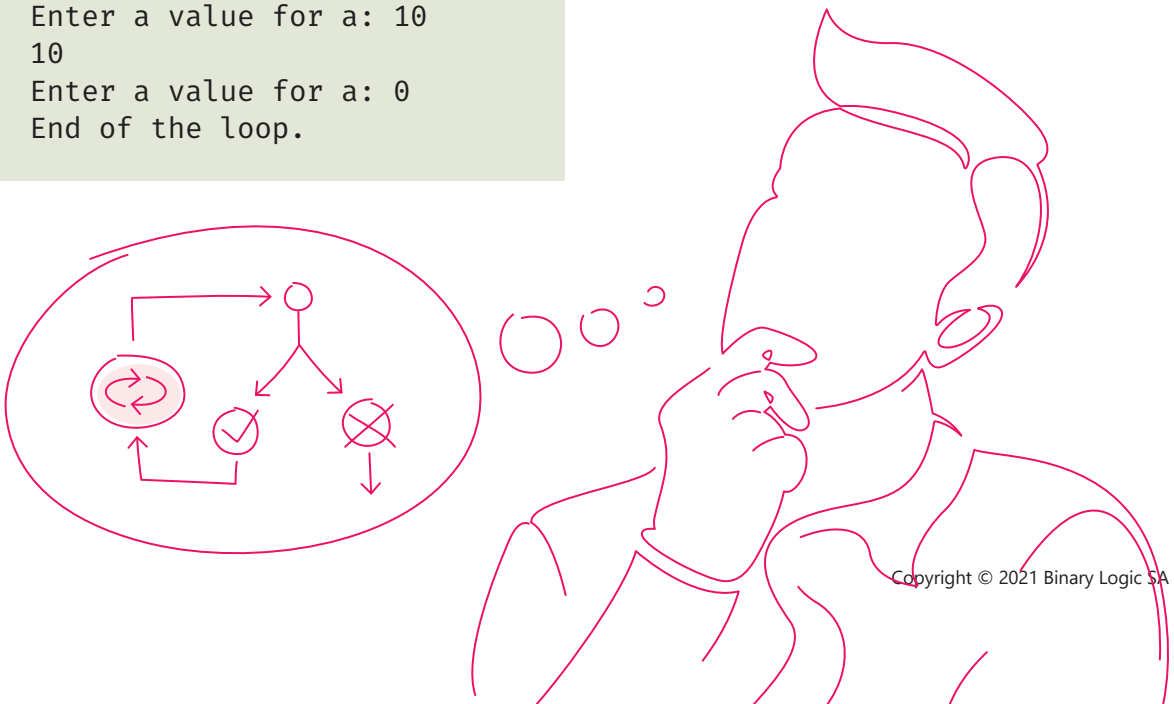
```
while condition:
    statements
```

The statements which are repeated have to be indented.

Let's look at an example with the **while** loop. In this example, the user enters a value for the variable **a**. The loop ends when the user enters **0** as the value for the variable **a**.

```
a=int(input("Enter a value for a: "))
while a!=0:
    print(a)
    a=int(input("Enter a value for a: "))
print("End of the loop.")
```

```
Enter a value for a: 5
5
Enter a value for a: 6
6
Enter a value for a: 10
10
Enter a value for a: 0
End of the loop.
```



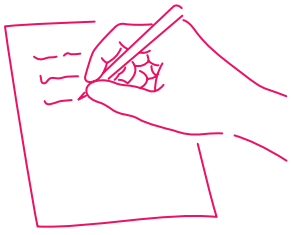
We can use the while loop to check a user's input into a variable.

Students letter grades

In this example, the program will ask the user to enter the grade of a student. This grade should be greater than or equal to **0** and less than or equal to **20**. If the user enters a value out of this range, the program will display an error message and ask the user to enter a valid grade.

```
#Students grades must be greater than or equal to 0
#and lower than or equal to 20
grade=int(input("Enter a student's grade: "))
while grade<0 or grade >20:
    print("Invalid grade, enter a grade between 1-20.")
    grade=int (input("Enter a valid grade: "))
print ("Your grade is: ", grade)
```

```
Enter a student's grade: 67
Invalid grade, enter a valid grade between 1-20.
Enter a grade: 18
Your grade is: 18
```



Try it out

Try out the following code and write down what appears on the screen.

```
i=1
while i<6:
  i=i+1
  if i == 3:
    print("Hello!")
  print(i)
```

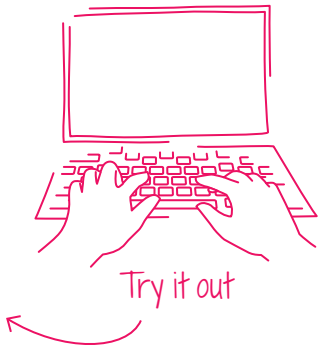
Infinite loop

If the condition of the **while** loop never becomes **False** you will end up with an infinite loop. An infinite loop is a loop which never ends.

When you use the **while** loop you should include a command, or a set of commands that will change the condition from **True** to **False**.

Try out the following code. What do you notice?

```
i=1
while i<6:
    print(i)
```



In the previous example, the value of the variable **i** does not change, so the program will be executed forever.

To stop the loop you must press Ctrl + C in the Python shell window.

Break Statement

Sometimes you want to terminate a loop before the condition becomes **False**. In such cases, you use a `break` statement. The `break` statement terminates the loop containing it. The program continues to the statement after the body of the loop. A `break` statement can also be used in the **for** loop.

```
while True:
    word=input("Type a word: ")
    if word=="stop":
        print("You used the break statement.")
        break
    print("Type a different word: ")
```

```
Type a word: this
Type a different word:
Type a word: that
Type a different word:
Type a word: stop
You used the break statement.
```

Go further!

There are usually many different ways to perform the same task.
One method is preferred over another based on several factors, the most important of which are the program's running speed and the required storage space. The programmer determines the best method.

1.3 Practice

1 How many times will the command `print()` be executed? Choose the correct answer:

```
for i in range (0,5,3):
   print(i)
```

- The program will not work because the syntax of the commands is incorrect.
- No message will be displayed on the screen because the condition is incorrect.
- The message "positive number" is displayed on the screen.

```
for i in range (10,1,-2):
   print(i)
```

- The program will not work because the syntax of the commands is incorrect.
- No message will be displayed on the screen because the condition is incorrect.
- The message "positive number" is displayed on the screen.

```
i=5
while i>1:
   print(i)
   i=i-1
```

- The program will not work because the syntax of the commands is incorrect.
- No message will be displayed on the screen because the condition is incorrect.
- The message "positive number" is displayed on the screen.

2 Write a program which will display the numbers 100, 95, 90, ..., 0 on the screen.

my code

DIGITAL Teens

can_code

with Python 2

Digital Teens is a graded Computing and ICT series that adopts an innovative project-based approach. Students understand computing concepts and develop their ICT skills through fun, real world scenarios and engaging activities.

Key features

- > Spiral presentation of Digital Literacy, Computer Science and ICT concepts
- > Clear step-by-step walkthroughs of operating systems and software applications
- > Comprehensive coverage of international curricula and exams
- > Effective integration of 21st century skills: collaboration, communication, creativity, critical thinking, problem-solving and decision-making
- > Extensive computational thinking support with supplementary material for Coding and Robotics for a variety of programming tools and educational robots
- > A wide range of online student resources provide flexibility and differentiation
- > Digital Teacher's Guides fully support novice and experienced teachers with step-by-step lesson plans

binarylogic
binarylogic.net

ISTE Standards Seal of Alignment Student Standards Jan 2021-23

ISBN: 978-960-698-467-9