



FAKULTAS
ILMU
KOMPUTER

Topic 3.2: Repetition

CSGE601020 - Dasar-Dasar Pemrograman 1

Lintang Matahari Hasani, S.Kom., M.Kom. | Dr.Eng. Lia Sadita, S.Kom., M.Eng.

Acknowledgement

This slide is an adapted version of 'Repetition' slides used in DDP1 Course (2020/2021) by **Hafizh Rafizal Adnan, M.Kom.**

Several materials are reused from 'Loops/Perulangan' slides used in Dasar-Dasar Pemrograman 1 dengan Python (CSGE601020/4 SKS) Course (<https://ocw.ui.ac.id/course/view.php?id=142>) by **Fariz Darari, Ph.D.** and 'Control (Branching and Loops)' slides by **Raja O. P. Damanik, M.Sc.** and **Dr. Eng. Lia Sadita**

Some of the design assets used in these slides were provided by ManyPixels under an nonexclusive, worldwide copyright license to download, copy, modify, distribute, perform, and use the assets provided from ManyPixels for free, including for commercial purposes, without permission from or attributing the creator or ManyPixels.

Copyright 2020 MANYPIXELS PTE LTD

Some additional contents, illustrations and visual design elements are provided by **Lintang Matahari Hasani, M.Kom.**



In this session, you will learn ...

`while` loop

`for` loop

`break` and `continue`

`range()` function

how to deal with infinite loop



Repeating Statements

Besides selecting which statements to execute, a fundamental need in a program is repetition

→ **repeat** a set of statements under some **conditions**

With both selection and repetition, we have the two most necessary programming ingredients



while loop

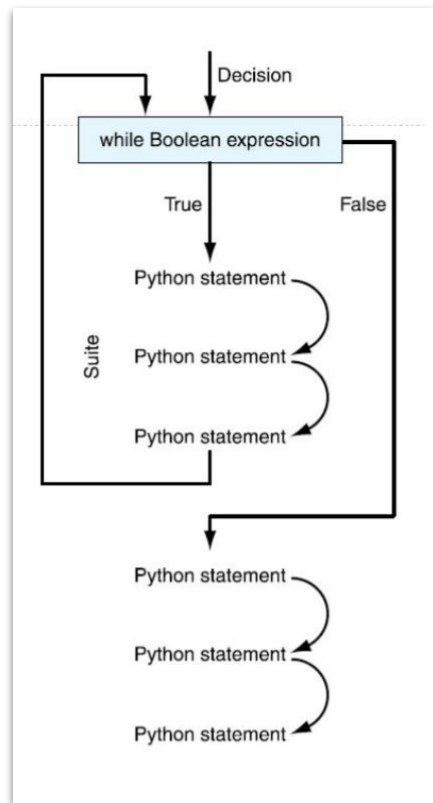
- Evaluates <condition> to a Boolean
- if <condition> is `True`, do all the steps inside the while code block
- check <condition> again
- repeat <condition> until <condition> is `False`

```
while <condition>:  
    #suite
```

Sample problem:

Suppose we want to print numbers from 5 to 1 (inclusive & descending)

```
n = 5  
while n > 0:  
    print(n)  
    n = n - 1
```



Developing while loop

Working with the loop control variable:

- Initialize the **variable**, typically outside of the loop and before the loop begins.
- The **condition statement** of the while loop involves a Boolean using the variable.
- **Modify the value** of the control variable **during the course of the loop**

```
n = 5
while n > 0:
    print(n)
    n = n - 1
```

controlling expression (condition)

block to be repeated

while loop: Some Simple Examples

```
n = 5
while n > 0:
    n = n - 1
    print(n)
```

VS

```
n = 5
while n > 0:
    print(n)
    n = n - 1
```

VS

```
n = 1
while n < 6:
    n = n + 1
    print(n)
```

Do these codes produce same outputs? why?

while loop with conditionals

```
n = 100
while n >= 0:
    if n % 2 == 0:
        print('Angka', n , 'adalah angka genap')
    n = n - 1
```


Triggering Question 1

Create a program to print odd numbers from 1 to 21 (inclusive) using while loop!

Hint: You may modify the codes in the previous slide ^^

Write your answer in the **Scele Forum**



```
1
3
5
7
9
11
13
15
17
19
21
```

Infinite while loop

while loop will repeat the statements in the suite while the Boolean is `True` (or its Python equivalent)

If the Boolean expression never changes during the course of the loop, the loop will continue forever (**infinite loop**).

- **while loop is an indefinite iteration:** The number of iterations is not explicitly stated, only on what condition the suite will be executed repeatedly
- **Be careful with infinite loop!**
- Make sure that our program terminates: The <condition> has to be false at some point; or a break statement has to be executed at some point.

Example:

```
# how to annoy a friend
n = 0
while n < 10:
    print('p')
```

What does this program do?

Triggering Question 2

Fix these while loops!

```
n = -100
while n > 100:
    print(n)
    n = n - 1
```

1

Suppose we want to print numbers from -100 to 100 (ascending, inclusive). How would you fix the above code?

```
n = 100
while n < 0:
    print(n)
    n = n - 1
```

2

Suppose we want to print numbers between 1 and 100 (ascending, not inclusive). How would you fix the above code?

Write your answer in the **comment section**



Issues in while loop

```
n = 10
while n < 0:
    n -= 1
    print('oraoraoraora')
```

Loop never starts:

The control variable is not initialized as you thought
(or perhaps you don't always want it to start)

```
n = 10
while n > 0:
    print('oraoraoraora')
```

Loop never ends:

The control variable is not modified during the loop
(or not modified in a way to make the Boolean come out False)



while loop with else (1)

while loop, oddly, can have an associated `else` suite

else suite is executed **when the loop finishes under normal conditions** – basically the last thing the loop does as it exits

```
while <condition>:  
    # suite  
else:  
    # suite  
  
# rest of the program
```

Example:

```
n = 7  
while n > 0:  
    n -= 1  
    print('oraoraoraora')  
else:  
    print('za warudo!')  
  
print('finish line')
```

What does this program do?

while loop with else (2)

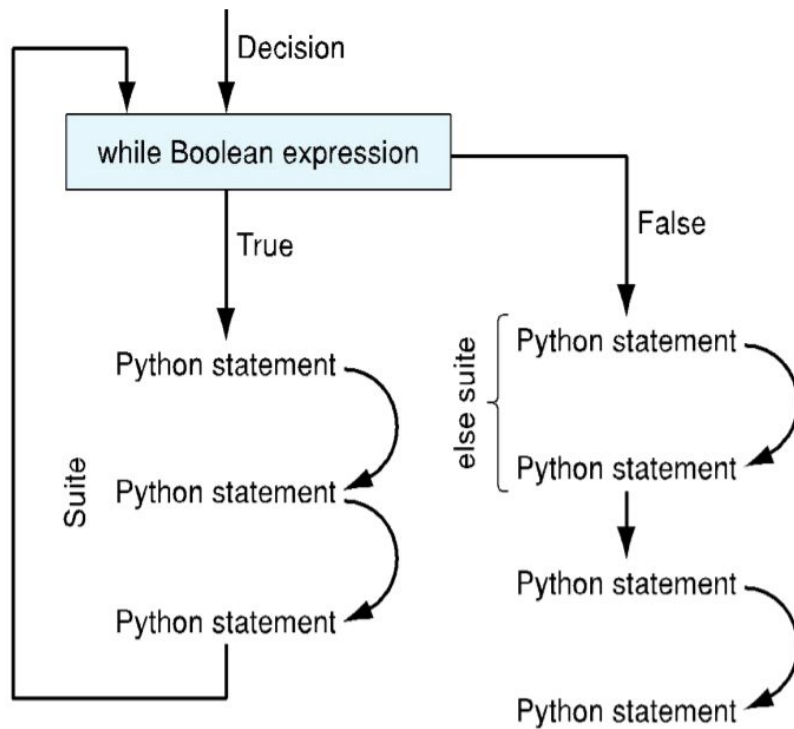


FIGURE 2.9 *while-else*.

Sample codes (1)

```
counter = 0

while counter < 3:
    print("Inside while with counter:", counter)
    counter = counter + 1
else:
    print("Inside else")
```

What does this program do?

Sample codes (2): Nested while

```
counter_i = 1
while counter_i < 4:
    counter_j = 0

    print('Iterasi outer loop ke-', i)

    while counter_j < 5:
        print(str(counter_i) + str(counter_j), end = ' ')
        counter_j = counter_j + 1

    print()
    counter_i = counter_i + 1
```

What does this program do?

Sentinel loop

Loop that is controlled by a sentinel (a particular value to terminate the loop).

Example: the following program will keep prompting an input from users until they type "quit"

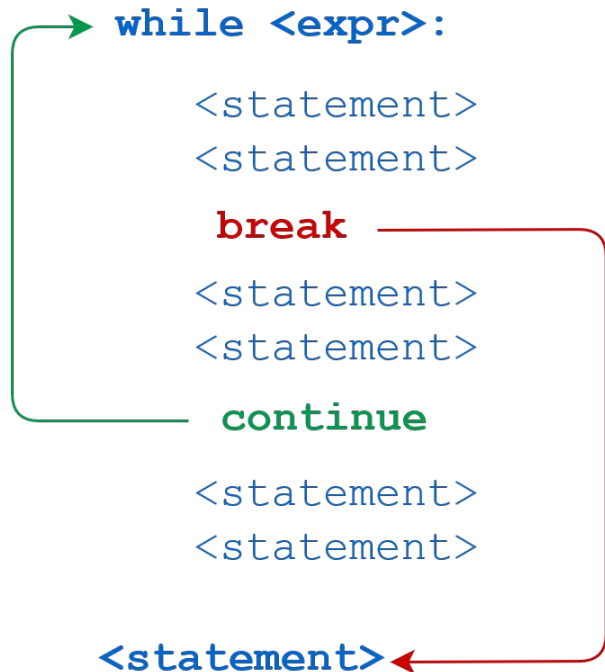
```
x = input("Enter something: ")
password = "quit"

while x != password:
    print("you entered ", x, ".", "Please enter the right password to exit this loop.")
    x = input("Enter something: ")

print("Congrats! Loop ended")
```

```
while <value> != <sentinel value>:
    # process value
    # get another value
```

break and continue statement



```
n = 5  
while n > 0:  
    n -= 1  
    if n == 2:  
        break  
    print(n)  
print('Loop ended.')
```

```
n = 5  
while n > 0:  
    n -= 1  
    if n == 2:  
        continue  
    print(n)  
print('Loop ended.')
```

break statement

A `break` statement in a loop, if executed, **end** the loop

- It **exits immediately, skipping whatever remains** of the loop as well as the else statement (if it exists) of the loop
- used for a non-normal exit (early exit) of the loop

```
counter = 0

while counter < 3:
    print("Inside while with counter:", counter)
    if counter == 1:
        break
    counter = counter + 1    # will not be executed after counter == 1
else:
    print("Inside else")    # will not be executed after counter == 1
```

continue statement (1)

- A continue statement, if executed **in a loop**, means to immediately **jump back to the top** of the loop and re-evaluate the conditional
- Any remaining parts of the loop are **skipped for one iteration** when the `continue` was executed

```
jumlah_genap = 0

while jumlah_genap < 3:
    angka = int(input("Masukkan angka genap: "))

    # Jika isian user angkanya ganjil, balik ke atas, nilai jumlah_genap tidak +1
    # n % 2 != 0 adalah bilangan ganjil (kalau dibagi 2 ada sisanya)
    if angka % 2 != 0:
        continue

    jumlah_genap += 1
    print("Jumlah sampai saat ini:", jumlah_genap)
```

continue statement (2)

- `while` loops are easiest to read when the **conditions of exit are clear**
- Excessive use of `continue` and `break` within a loop suite make it **more difficult to decide** when the loop will exit and what parts of the suite will be executed each loop.

Use them wisely ^^

Triggering Question 3

Guess the output!

Write your answer in the **Comment Section**



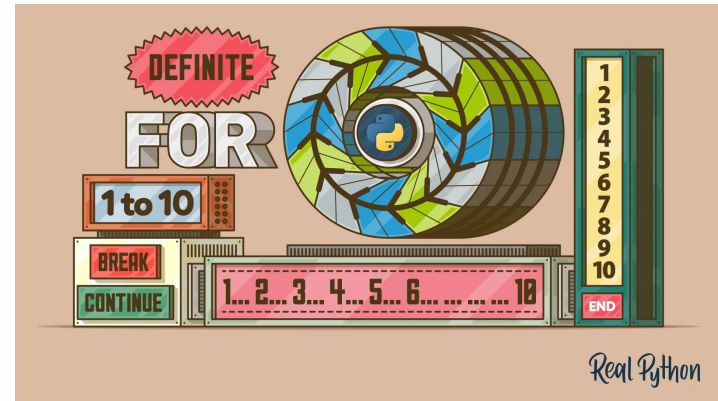
```
n = 7
while n > 0:
    n -= 1
    print('oraoraoraora')
    if n == 4:
        break
    else:
        continue
else:
    print('mudamudamudamuda!')

print('finish line')
```

while and for statements



The **while** statement is more general. It repeats a set of statements while **some condition is True**.



The **for** statement is useful for iteration, **moving through all the elements of data structure**, one at a time.

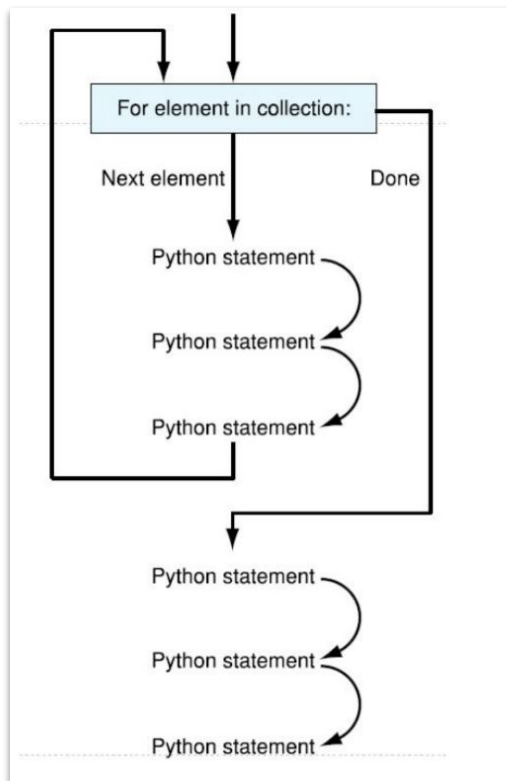
for loop

One of Python's strengths is its rich set of built-in data structures
The `for` statement **iterates through each element of a collection** (list, etc.)

```
for <element> in <collection>:  
    #suite
```

```
for <variable> in range(<num>):  
    #suite
```

- Each time through the loop, `<variable>` takes a value from range
- Do all the steps in the for-suite
- gets the next value
- Easier to guarantee termination! :D

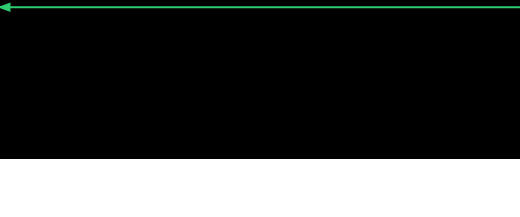


An Example of for Loop

```
# prints each letter in string 'Python'
for letter in 'Python':
    print('Current Letter :', letter)

# list of fruits
fruits = ['banana', 'apple', 'mango']

for fruit in fruits: # Second Example
    print('Current fruit :', fruit)
```



This is a list, a collection and an **iterable**.
Don't worry about this, we will learn about this later ^^

range () function (1)

to generate a **sequence of integers**

The range function takes 3 arguments:

- the **beginning** of the range. Assumed to be 0 if not provided
- the **end** of the range, but **not inclusive** (up to but not including the number) ← **Required**
- the **step** of the range. Assumed to be 1 if not provided

If **only one** argument is provided, it is assumed to be the **end value**
ex:

```
for x in range (3):  
    # this loop will have 2 iterations  
    # The beginning will be assumed to be 0  
    # The step will be assumed to be 1
```

```
for x in range (2, 100, 3):  
    # do something 32 times
```

```
for x in range (0, 100):  
    # do something 100 times
```

```
for x in range (20, 100):  
    # do something 79 times
```

range () function (2)

A range is also called iterable, because we can iterate through a range.

For example:

```
for num in range(1, 5):  
    print(num)
```

- `range(1, 5)` represents the **sequence** 1, 2, 3, 4
- `for` loop assigns `num` to each of the values in the sequence, **one at a time, in sequence**
- **prints each number** (one number per line)

range () function (3)

- `range(<stop>)`
The same as `range(0, stop)`
- `range(<start>, <stop>)`
The same as `range(start, stop, 1)`
- `range(<start>, <stop>, <step>)`
Take value start, then start+step, and keep adding step as long as the value is < stop

```
for x in range (12):  
    # do something 11 times
```

```
for x in range (0, 12):  
    # do something 11 times
```

```
for x in range (12):  
    # do something 11 times
```

Sample codes

```
mysum = 0
for i in range(7, 10):
    mysum += i
    print(mysum)
```

```
mysum = 0
for i in range(5, 11, 2):
    mysum += i
    print(mysum)
```

What do these programs do?

```
mysum = 0
for i in range(10, 3, -2):
    mysum += i
    print(mysum)
```

Triggering Question 4

**Are while loop
and for loop convertible?
Why?**

Write your answer in the **comment section**



In most cases, while loop and for loop are both convertible

```
n = int(input('masukkan angka:'))  
  
while(n > 0):  
    print('*')  
    n -= 1
```



```
n = int(input('masukkan angka:'))  
  
for x in range (n, 0, -1):  
    print('*')
```

while loop VS for loop

- Usually, people use `for` loop when they know how many repetitions they need to do. Say you want to print "I love programming" 10 times, then use `for` loop.
- People use `while` loop when they want repetitions to happen as long as a certain condition holds or as long as a certain condition has not happened yet. Say, when keep asking a password while the input password is still incorrect.
- While-loop may suffer from infinite loop terribly. Programmers must make sure the while-loop terminates (will the condition be false at some point?)



Triggering Question 5

Classic Triangle Star

Create a program that receive an integer x input and print right triangle with height x.

You may use `while` or `for` loop

Hint: You may use nested loop ^-^

3

```
*  
**  
***
```

4

```
*  
**  
***  
****
```

5

```
*  
**  
***  
****  
*****
```

Write your answer in the **Scele Forum**



Review Questions

Explain how `for` and `while` loops work.

When should we use loop using `for` or `while` statements?

Why infinite loop happens? How can we prevent it?





UNIVERSITAS
INDONESIA
Terbuka, Mandiri, Berkualitas

FAKULTAS
ILMU
KOMPUTER

Q&A Session

