CONDITIONALS AND LOOPS (Lanjutan)

Outline

- More on Variable Assignments
- More on Conditionals
- Iterations

More on Assignments

Remember Assignments?

- Format: `lhs = rhs`
- Behavior:
  - expression in the **rhs** is evaluated producing a value
  - the value produced is placed in the location indicated on the **lhs**

Can do multiple assignments

```
a_int, b_int = 2, 3
```

first on right assigned to first on left, second on right assigned to second on left

```
print(a_int, b_int)   # prints 2 3
```

```
a_int,b_int = 1,2,3
```
→ Error

counts on lhs and rhs must match

traditional swap

- Initial values: `a_int= 2, b_int = 3`
- Behavior: swap values of `X` and `Y`
  - introduce extra variable `temp`
    - `temp = a_int` # save a_int value in temp
    - `a_int = b_int` # assign a_int value to b_int
    - `b_int = temp` # assign temp value to b_int

Swap using multiple assignment

- `a_int, b_int = 2, 3`
- `print(a_int, b_int)` # prints 2 3

- `a_int, b_int = b_int, a_int`
- `print(a_int, b_int)` # prints 3 2

- remember, evaluate all the values on the **rhs** first, then assign to variables on the **lhs**

Chaining for assignment

Unlike other operations which chain left to right, assignment chains right to left

```
a_int = b_int = 5
print(a_int, b_int) # prints 5  5
```

More on Conditionals

Compound Statements

- Compound statements involve a set of statements being used as a group
- Most compound statements have:
  - a header, ending with a `:` (colon)
  - a suite of statements to be executed
- **`if, for, while`** are examples of compound statements

We have seen 2 forms of selection

```
1). if boolean-expression:
        suite

2). if boolean-expression:
        suite
    else:
        suite
```

Python Selection, Round 3

```
if boolean-expression1:
      suite1
elif boolean-expression2:
      suite2
#(as many elif's as you want)
…
else:
      suite_last
```

`if, elif, else`, the process

- Evaluate Boolean expressions until:
  - the Boolean expression returns `True`
  - none of the Boolean expressions return `True`
- if a boolean returns `True`, run the corresponding suite. Skip the rest of the `if`
- if no boolean returns `True`, run the `else` suite, the default suite

Latihan Coding: Nilai Alpro

Nilai	Min	Maks
A	85	100
A-	80	85
B+	75	80
B	70	75
B-	65	70
C+	60	65
C	55	60
D	40	55
E	0	40

Iterations

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` 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 (or some range), one at a time.

`while` loop

- Top-tested loop (pretest):
  - test the boolean before each iteration of the loop (*incl. the first iteration*)

```
while boolean expression:
    suite
```

`while` loop

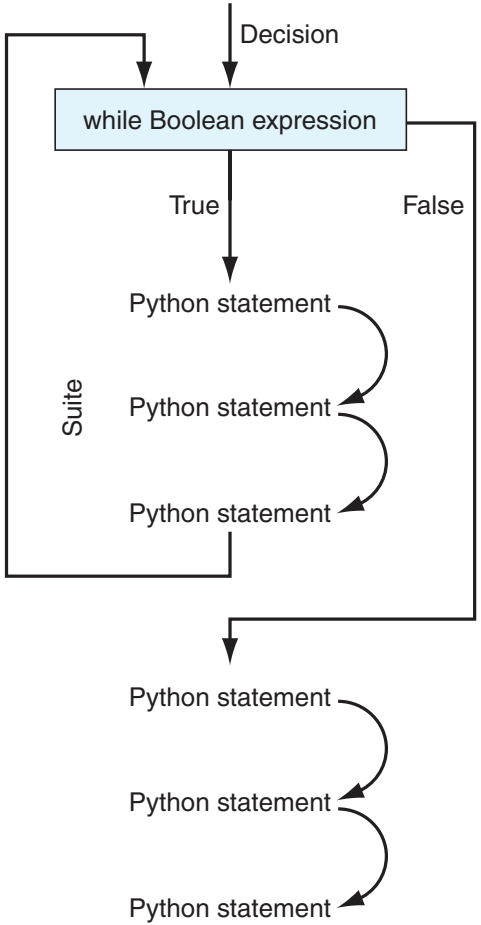


FIGURE 2.4 ***while*** loop.

repeat while the boolean is true

- 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.

Live Coding: Kura berputar

```
import turtle

t = turtle.Screen()
kura = turtle.Turtle()
kura.shape("turtle")

jumlah_lingkaran = t.numinput("Contoh Program", "Jumlah lingkaran:")

counter = 0

while counter < int(jumlah_lingkaran):
    kura.circle(20)
    kura.forward(40)
    counter = counter + 1
kura.write("        ????")
```

Developing a `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

Issues:

- Loop never starts:

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

- 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`)

Live Coding: Displaying Matrix (Using Nested `while`)

```
counter_i = 0
while counter_i < 3:
    counter_j = 0
    while counter_j < 5:
        print(str(counter_i)+str(counter_j), end = ' ')
        counter_j = counter_j+1
    print()
    counter_i = counter_i+1
```

`for` iteration

- 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.) or some range

```
for element in collection:
    suite
```

or

```
for i in range(N):
    suite
```

`for` loop

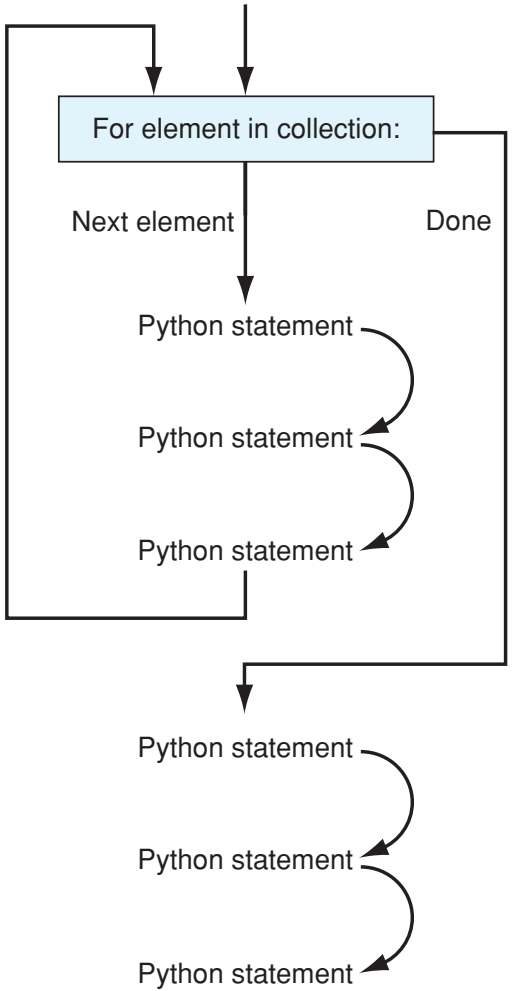


FIGURE 2.5 Operation of a ***for*** loop.

For loop for collections (list, etc)

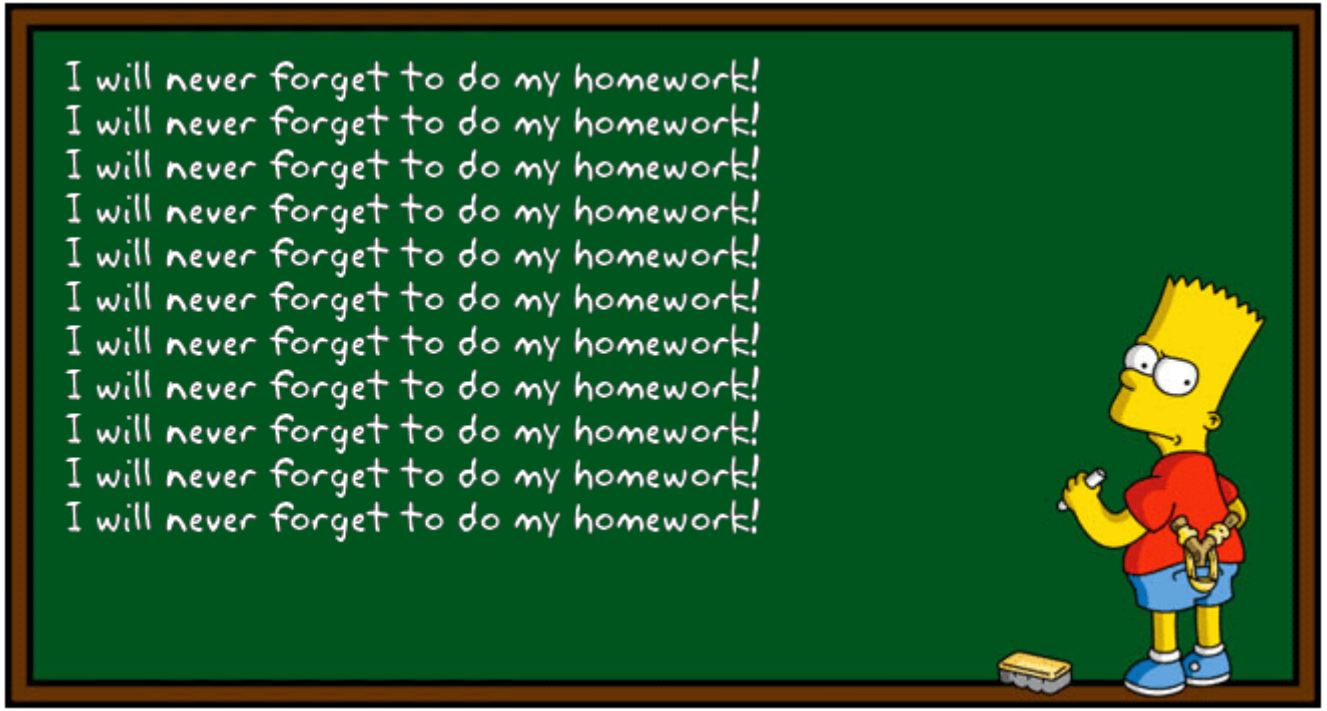
```
for el in [5,1,7,1]:
  print(el)
```

```
5
1
7
1
```

For loop for collections (list, etc)

```
s = 0
for el in [5,1,7,1]:
  s += el
print(s)
```

Suppose you have to do the same thing N times



`Range` function

- 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 arg provided, assumed to be the **end value**

For loop: When knowing how many times the loop will execute

```
for i in range(11):
  print("I will never forget to do my homework!")
```

```
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
I will never forget to do my homework!
```

For loop: When knowing how many times the loop will execute

```
for i in range(11):
  print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

For loop: When knowing how many times the loop will execute

```
for i in range(6,11):
  print(i)
```

```
6
7
8
9
10
```

For loop: When knowing how many times the loop will execute

```
for i in range(6,11,2):
  print(i)
```

```
6
8
10
```

Quiz time: What's the output?

```
for i in range(5):
  if i % 2 == 0:
    print("Genap")
  else:
    print("Ganjil")
```

Quiz time: What's the output?

```
for i in range(5):
  if i % 2 == 0:
    print("Genap")
  else:
    print("Ganjil")
```

```
Genap
Ganjil
Genap
Ganjil
Genap
```

Live Coding: Bermain dengan range

```
for i in range(10):
    print(i, end=" ")

print()

for i in range(1,7):
    print(i, end=" ")

print()

for i in range(0,30,5):
    print(i, end=" ")

print()

for i in range(5,-5,-1):
    print(i, end=" ")
```

`while` loop, round two

- 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` with `else`

```
while booleanExpression:
      suite
      suite
else:
      suite
      suite
rest of the program
```

`while-else` loop

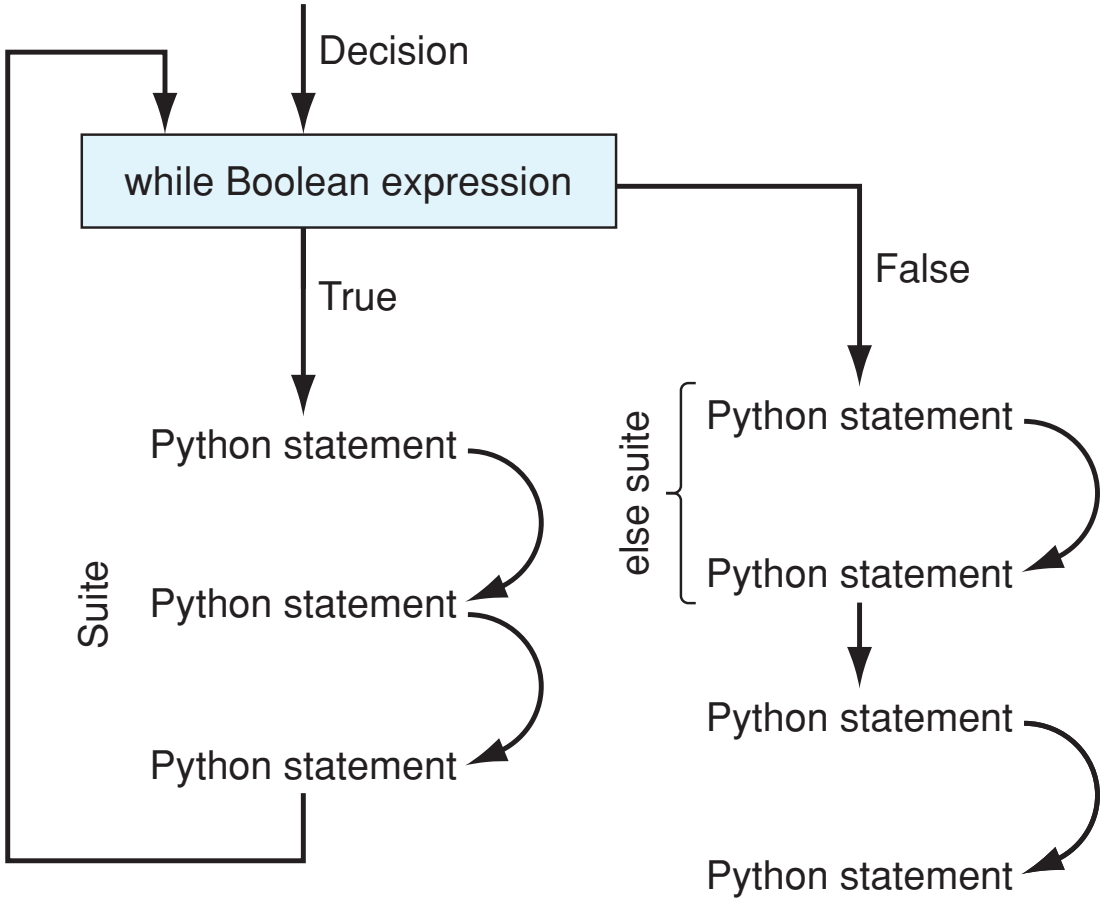


FIGURE 2.9 ***while-else.***

Live Coding: `while-else`

```
counter = 0

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

`break` statement

- A `break` statement in a loop, if executed, exits 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

Live Coding: `while-else-break`

```
counter = 0

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

Live Coding: Guess a number

```
import random

number = random.randint(0,10) # get random number between 0-10
guess = int(input("Guess a number (0-10): "))

while 0 <= guess <= 10:
    if guess > number:
        print("Too high")
    elif guess < number:
        print("Too low")
    else:
        print("You guessed it! The number was", number)
        break
    guess = int(input("Guess (again) a number (0-10):"))
else:
    print("Program quit, input is not right!")
```

`continue` statement

- 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 the one iteration when the continue was executed

Live Coding: Masukkan 3 angka genap

```
jumlah_genap = 0

while jumlah_genap < 3:
    angka = int(input("Masukkan angka genap: "))
    if angka % 2 != 0:
        continue
    jumlah_genap += 1
    print("Jumlah sampai saat ini:", jumlah_genap)
```

change in control: `break` and `continue`

- 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 judiciously.~~ It is highly recommended to not use this at all (provide better `while` or `loop` structure)

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()
while x!="quit":
    print("you entered ", x)
    x=input()
```

General form:

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

While overview

```
while test1:
     # statement_list_1
     if test2:
          break          # Exit loop now; skip else
     if test3:
          continue       # Go to top of loop now
     # more statements
else:
     # statement_list_2   # If we didn't hit a 'break'
```

`For` overview

- just like the while statement, for statement can also support `else`, `continue`, and `break`.

```
for target in object:
     # statement_list_1
     if test1:
          break              # Exit loop now; skip else
     if test2:
          continue           # Go to top of loop now
     # more statements
else:
     # statement_list_2      # If we didn't hit a 'break'
```

Equivalence of `while` and `for`

- It is possible to write a `while` loop that behaves like a `for` loop.

Exercise: Write an equivalent form of the following code using while loop!

```
for i in range(5):
    print(i)
```

Practice

Live Coding: Mencari maksimum dari 3 angka input

Live Coding: Mencari maksimum dari 3 angka input

```
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))

if x >= y and x >= z:
  print("Max:", x)
elif y >= x and y >= z:
  print("Max:", y)
else:
  print("Max:", z)
```

Live Coding: Berapa angka yang sama dari 3 angka

```
a = int(input())
b = int(input())
c = int(input())

# masukkan kode di sini
```

The program must print one of the numbers: 3 (if all are same), 2 (if two of them are equal to each other and the third is different) or 0 (if all numbers are different).

Live Coding: Berapa yang sama dari 3 angka

```
a = int(input())
b = int(input())
c = int(input())

if a == b == c:
    print(3)
elif a == b or a == c or b == c :
    print(2)
else:
    print(0)
```

The program must print one of the numbers: 3 (if all are same), 2 (if two of them are equal to each other and the third is different) or 0 (if all numbers are different).

Live Coding:
Diberikan angka N, hitung 1 + 2 + ... + N

Live Coding:
Diberikan angka N, hitung 1 + 2 + ... + N (Oops!)

```
n = int(input())

total = 0
for i in range(n):
  total = total + i

print(total)
```

Live Coding:
Diberikan angka N, hitung 1 + 2 + ... + N

```
n = int(input())

total = 0
for i in range(n+1):
  total = total + i

print(total)
```

Live Coding:
Diberikan angka N, hitung N! (= 1 * 2 * ... * N) (Oops!)

```
n = int(input())

total = 1
for i in range(n+1):
  total = total * i

print(total)
```

Live Coding:
Diberikan angka N, hitung N! (= 1 * 2 * ... * N)

```
n = int(input())

total = 1
for i in range(1, n+1):
  total = total * i

print(total)
```

Live Coding: Diberikan angka N, gambar piramida setengah seperti ini:

```
          * # sebanyak 1
         ** # sebanyak 2
          .
          .
          .
 ********** # sebanyak N-1
********** # sebanyak N
```

Contoh apabila N = 5:

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

Hint: Suatu string dapat dicetak secara berulang menggunakan *
Contoh: "a"*3 = "aaa"

Live Coding: Diberikan angka N, gambar piramida setengah!

```
n = int(input())

for i in range(1,n+1):
  print(" "*(n - i) + "*"*(i))
```