Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas

INF 100 – Introduction to Programming

Repetition statements:

WHILE

Programs without repetition...

- The computer executes sequentially each instruction of the program, at most once.
- Some instructions may not be executed because of conditional commands.

Example

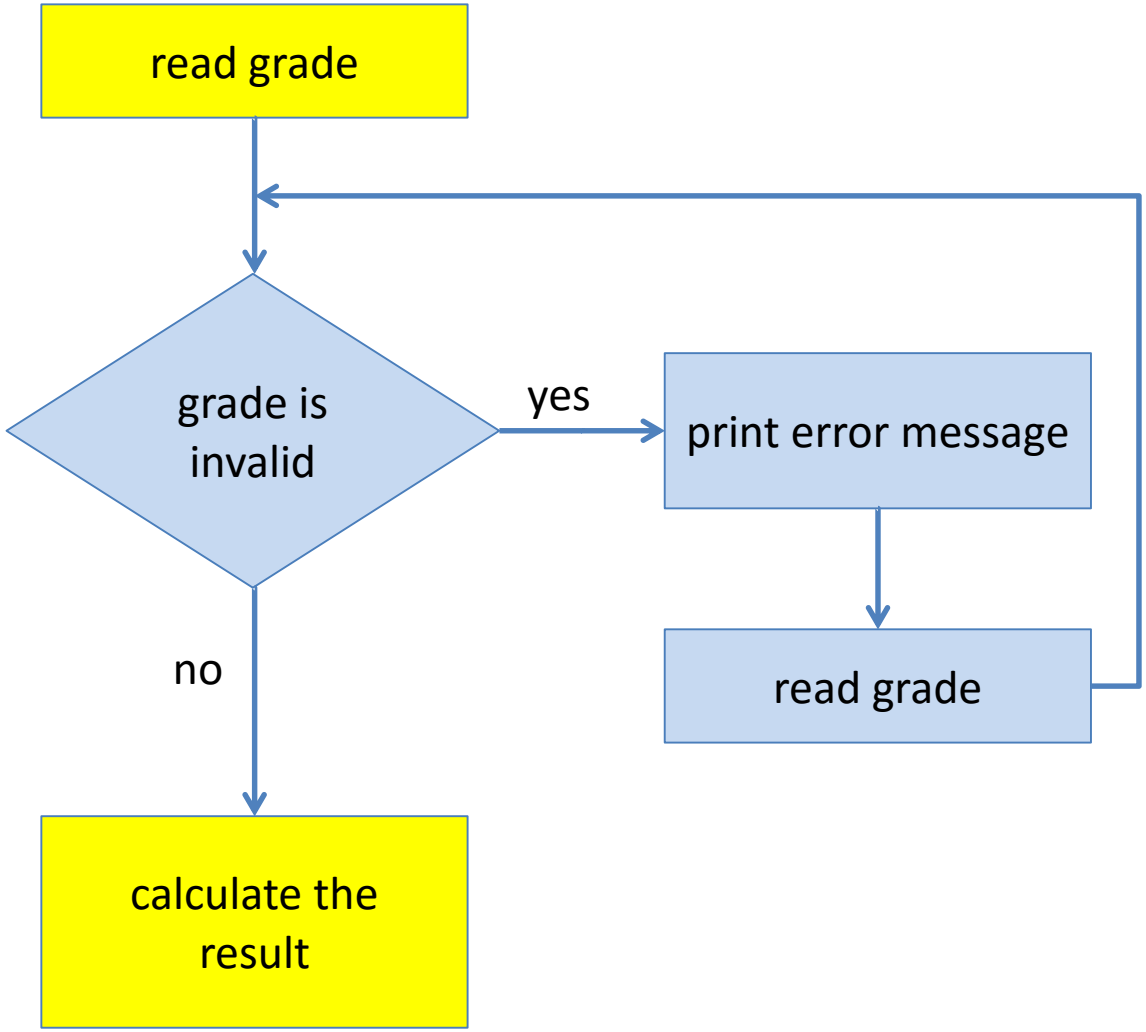
```
grade = float (input("Type the grade: "))
if grade < 0 or grade > 100:
    print("Invalid value")
elif grade >= 60:
    print("Approved")
elif grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

- Problem: ensure that the user will eventually type a valid value. The program must request the user to retype it until the value is valid.

Example of execution

```
Type the grade: 120
Invalid value
Type the grade: -5
Invalid value
Type the grade: 70
Approved
```

Solution with flowchart



Algorithm – using while statement

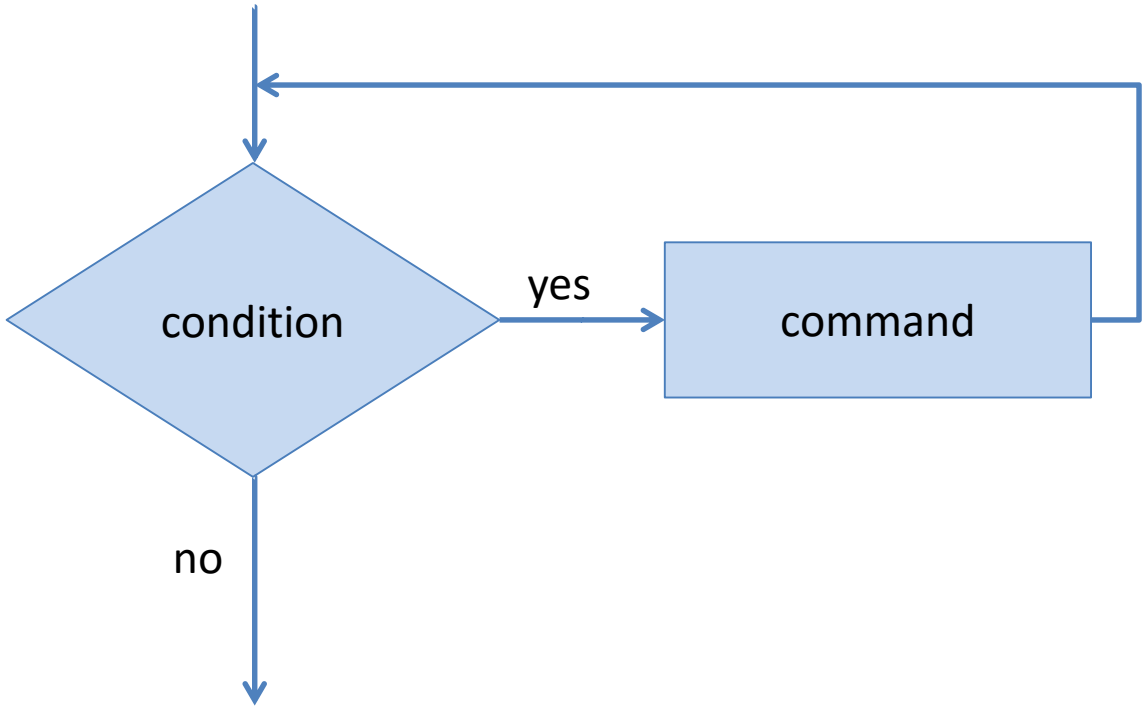
```
read grade
while grade is invalid
    print error message
    read grade
process grade and print result
```

Python - while statement

- Syntax:

```
while condition :
    command
```

- Semantics:



Solution in Python

```
grade = float (input("Type the grade: "))
while grade < 0 or grade > 100:
    print("Invalid value")
    grade = float (input("Type the grade: "))

if grade >= 60:
    print("Approved")
elif grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

(program **grade.py**)

Command **break**

- The command **break** finishes the current repetition. Example:

```
while True:
    n = int(input("How many values? "))
    if n > 0:
        break
    print ("Number of values must be positive!")
```

Command **break**

- The command **break** finishes the current repetition. Example:

```
while True:
    n = int(input("How many values? "))
    if n > 0:
        break
    print ("Number of values must be positive!")
```

Means: “repeat forever”

If the value typed is positive, then the repetition is interrupted

Command **break**

- The command **break** finishes the current repetition. Example:

```
while True:
    n = int(input("How many values? "))
    if n > 0:
        break
    print ("Number of values must be positive!")
```

- The use of the command **break** is not always advised, unless the benefits for the code are clear.

EXERCISE:
- CHANGE THE CODE OF PROGRAM GRADE.PY, NOW USING BREAK
- TEST THE PROGRAM WITH SOME INVALID AND VALID DATA AS INPUT

Solution in Python – version 1

```
while True:
    grade = float (input("Type the grade: "))
    if 0 <= grade <= 100:
        break
    else:
        print("Invalid value")

if grade >= 60:
    print("Approved")
elif grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

Solution in Python – version 2

```
while True:
    grade = float (input("Type the grade: "))
    if 0 <= grade <= 100:
        break
    print("Invalid value")

if grade >= 60:
    print("Approved")
elif grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

Comparing with version 1, this version does not use “else”. Try to understand why, IN THIS CASE, an “else” clause is not necessary.

Solution in Python – version 3

```
while True:
    grade = float (input("Type the grade: "))
    if grade < 0 or grade > 100:
        print("Invalid value")
    else:
        break

if grade >= 60:
    print("Approved")
elif grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

Comparing with version 1 and 2, this version tests the opposite condition.

Repeating commands a number of times

- In Python, there is no specific command that indicates other commands must be repeated a given number of times.
- In some algorithms, such construction could be useful. Examples:

```
repeat 4 times:
    command1
```

```
read k
repeat k times:
    command2
```

Repeating commands a number of times

- Ordinary repetition commands such as WHILE can be used to repeat other commands, given a number of times. Examples:

```
i = 1
while i <= 4:
    command1
    i += 1
```

```
read k
while k > 0:
    command2
    k -= 1
```

(supposing `command`$_1$ and `command`$_2$ do not change the values of variables `i` and `k`)

Exercise

- How many times **command$_1$** will be executed?

```
i = 1
while i < 4:
    command1
    i += 1
```

```
i = n  # n already defined
while i > 0:
    command1
    i -= 1
```

```
j = 0
while j < 4:
    command1
    j += 1
```

```
i = n  # n already defined
while i > 0:
    command1
    i += 1
```

Assignment #4 revisited

Exercise: simplify the code using repetition commands.

```
n = 0    # stores number of valid weights
sum = 0  # stores sum of valid weights

weight = float (input("weight of capybara 1: "))
if weight > 0:
    sum += weight
    n += 1
else:
    print("Error!")

weight = float (input("weight of capybara 2: "))
if weight > 0:
    sum += weight
    n += 1
else:
    print("Error!")

weight = float (input("weight of capybara 3: "))
if weight > 0:
    sum += weight
    n += 1
else:
    print("Error!")

weight = float (input("weight of capybara 4: "))
if weight > 0:
    sum += weight
    n += 1
else:
    print("Error!")

if n == 0:
    print("No valid values.")
else:
    print("Average = ", sum/n )
```

Template for solution

Create variable “i “ to control the number of repetitions

```
n = 0     # stores number of valid weights
sum = 0   # stores sum of valid weights

while

    weight = float (input("weight of capybara " + str(i) + ": "))
    if weight > 0:
        sum += weight
        n += 1
    else:
        print("Error!")

if n == 0:
    print("No valid values.")
else:
    print("Average = ", sum/n )
```

Write condition to control the number of repetitions

Update the variable that controls the number of repetitions

Solution

```
n = 0     # stores number of valid weights
sum = 0   # stores sum of valid weights

i = 1
while i <= 4:
    weight = float (input("weight of capybara " + str(i) + ": "))
    if weight > 0:
        sum += weight
        n += 1
    else:
        print("Error!")
    i += 1

if n == 0:
    print("No valid values.")
else:
    print("Average = ", sum/n )
```

Ensure user will type 4 valid values

```
n = 0     # stores number of valid weights
sum = 0   # stores sum of valid weights

while n < 4:
    weight = float (input("weight of capybara " + str(n+1) + ": "))
    if weight > 0:
        sum += weight
        n += 1
    else:
        print("Error!")

print("Average = ", sum/4 )
```

Average of a sequence of values

- Problem: write a program that reads a sequence of values and then calculates the average.
- The program does not know in advance how many values will be typed by an user.
- How to decide when the user has finished typing the sequence of values?

Average of a sequence of values

- Approach (1): before start reading the values, the program asks the user the number of values that will be typed.

- Sample execution:

```
How many values? 3
Type a value: 7
Type a value: 18
Type a value: 5
Average = 10
```

Python template for approach (1)

```
n = int(input("How many values? "))
i = n
# other initializations

while i > 0:
    value = float(input("Type a value: "))
    # process value
    i -= 1

# process the results
```

Average with approach (1)

```
n = int(input("How many values? "))
i = n
sum = 0

while i > 0:
    value = float(input("Type a value: "))
    sum += value
    i -= 1

avg = sum / n
print("Average = ", avg)
```

Improving the solution...

```
n = int(input("How many values? "))
while n <= 0:
    print ("Number of values must be positive!")
    n = int(input("How many values? "))
i = n
sum = 0

while i > 0:
    value = float(input("Type a value: "))
    sum += value
    i -= 1

avg = sum / n
print("Average = ", avg)
```

(program **average1a.py**)

Average of a sequence of values

- Approach (2): Define a “flag”, a value to end the sequence. For example, if only positive values are allowed, then the program may stop reading when the user types a negative value.
- Sample execution:

```
Type a value (<0 to finish): 7
Type a value (<0 to finish): 18
Type a value (<0 to finish): 5
Type a value (<0 to finish): -1
Average = 10
```

Python template for approach (2)

```
value = float(input("Type a value: "))
# other initializations

while value is not the flag:
    # process value
    value = float(input("Type a value: "))

# process the results
```

Average with approach (2)

```
value = float(input("Type a value: "))
n = 0
sum = 0

while value >= 0:
    sum += value
    n += 1
    value = float(input("Type a value: "))

avg = sum / n
print("Average = ", avg)
```

(program **average2.py**)

Improving the solution...

```
value = float(input("Type a value: "))
n = 0
sum = 0

while value >= 0:
    sum += value
    n += 1
    value = float(input("Type a value: "))

if n == 0:
    print("No values typed")
else:
    avg = sum / n
    print("Average = ", avg)
```

(program **average2a.py**)

Approach (2) using BREAK

```
# initializations

while True:
    value = float(input("Type a value: "))
    if value is the flag:
        break
    # process value

# process the results
```

Approach (2) using BREAK

```
n = 0
sum = 0

while True:
    value = float(input("Type a value: "))
    if value < 0:
        break
    sum += value
    n += 1

avg = sum / n
print("Average = ", avg)
```

(program **average3.py**)