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

INF 100 – Introduction to Programming

Conditional commands

Universidade Federal de Viçosa

Problem for motivation

- Problem: write a program that reads the coefficients a, b and c of a quadratic equation

$$ax^2 + bx + c = 0$$

and then calculates and prints the roots of this equation.

- Exercise: analyze the problem and build an initial algorithmic solution. Initially, you can suppose that $a>0$ and that the equation has real roots.

Initial solution

- For now, suppose that $a>0$ and that the equation has real roots.

- Algorithm:

```
read a, b, c
$\Delta \leftarrow b^2 - 4ac$
x1 ← $\frac{-b+\sqrt{\Delta}}{2a}$
x2 ← $\frac{-b-\sqrt{\Delta}}{2a}$
print x1
print x2
```

Exercise

```
read a, b, c
$\Delta \leftarrow b^2 - 4ac$
x1 ← $\frac{-b+\sqrt{\Delta}}{2a}$
x2 ← $\frac{-b-\sqrt{\Delta}}{2a}$
print x1
print x2
```

- Translate the algorithm to Python.

- For calculating square root, you can use the operator **, raising a value to 0.5

INF 100 – Introdução à Programação

Solution: program in Python

```
print("Quadratic equation ax² + bx + c = 0")
print("Type the coefficients:")
a = float (input("a = "))
b = float (input("b = "))
c = float (input("c = "))
delta = b*b - 4*a*c
x1 = (-b + delta**0.5) / (2 * a)
x2 = (-b - delta**0.5) / (2 * a)
print("x1 = ", x1 )
print("x2 = ", x2 )
```

Dealing with special situations

What to do in the following situations?

- $a = 0 \ldots ?$
- $\Delta < 0 \ldots ?$

Dealing with special situations

What to do in the following situations?

- $a = 0$ (it is not a quadratic equation)
- $\Delta < 0$ (equation has no real roots)

- It is necessary to use structures that allow the execution of commands under certain conditions!

Conditional commands

In algorithms, we can write:

```
if condition then
   $command_1$
   $command_2$
   ...
```

meaning that the listed commands will only be executed if the given condition is true.

Conditional commands

In algorithms, we can write:

```
if condition then
    $command_1$
    $command_2$
    ...
```

Other commands may follow the conditional command IF.
How to define which commands depend on the condition?
Options:
- **use indentation**;
- use explicit terminators.

Conditional commands

In algorithms, we can write:

```
if condition then
    $command_1$
    $command_2$
    . . .
```

OR

```
if condition then
    $command_1$
    $command_2$
    . . .
endif
```

Other commands may follow the conditional command IF.
How to define which commands depend on the condition?
Options:
- use indentation;
- **use explicit terminators**.

Conditional commands

- Compare the following pieces of algorithms:

```
if condition then
   command$_1$
   command$_2$
command$_3$
```

```
if condition then
   command$_1$
command$_2$
command$_3$
```

- Considering that indentation is used to define the dependence on the conditional command, then the two algorithms have a different semantics - what is the difference???

Conditional commands

- Compare the following pieces of algorithms:

```
if condition then
   $command_1$
   $command_2$
$command_3$
```

```
if condition then
   $command_1$
$command_2$
$command_3$
```

- In the first case:
  - 2 commands depend on the condition.
  - *command*$_3$ is always executed, regardless of the evaluation of the condition.

Conditional commands

- Compare the following pieces of algorithms:

```
if condition then
   $command_1$
   $command_2$
$command_3$
```

```
if condition then
$command_1$
$command_2$
$command_3$
```

- In the second case:
  - *command*$_1$ depends on the condition.
  - *command*$_2$ and *command*$_3$ are always executed, regardless of the evaluation of the condition.

Conditional commands

- Below we use the explicit terminator “endif” to define the dependence on the conditional commands , instead of considering indentation.

```
if condition then
   command$_1$
   command$_2$
endif
command$_3$
```

```
if condition then
   command$_1$
endif
command$_2$
command$_3$
```

Conditional commands

- When writing algorithms, you can use either indentation or explicit terminators to define dependence on commands.
- Some programming languages use indentation (e.g. Python) and other languages use explicit terminators (C, C++, Java...).

Conditional commands

An extended version of the conditional command is:

```
if condition then
   command$_1$
else
   command$_2$
```

meaning that, if the given condition is true, ***command***$_1$ will be executed;
otherwise, ***command***$_2$ will be executed.

Syntax in Python

Algorithm

```
if condition then
   command_1
```

```
if condition then
   command_1
else
   command_2
```

Python

```
if condition:
   command_1
```

```
if condition:
   command_1
else:
   command_2
```

Operators for comparison

Operator	Equivalent in Python
=	==
≠	!=
>	>
<	<
≥	>=
≤	<=

Translating to Python

```
if a ≠ 0 then
   b ← 1
   c ← 2
d ← 3
```

→

```python
if a != 0:
   b = 1
   c = 2
d = 3
```

```
if a ≥ 0 then
   b ← 1
else
   c ← 2
   d ←3
```

→

```python
if a >= 0:
   b = 1
else:
   c = 2
   d = 3
```

Exercise

- What will be printed by the following Python code?

```
a=2
b=1
if a < b:
   a=3
else:
   b=0
   a=4
print( a, b )
a=1
b=2
if a < b:
   a=3
else:
   b=0
a=4
print( a, b )
```

Exercise – extending the algorithm

```
read a, b, c
Δ ← $b^2 - 4ac$
x1 ← $\frac{-b+\sqrt{\Delta}}{2a}$
x2 ← $\frac{-b-\sqrt{\Delta}}{2a}$
print x1
print x2
```

- Extend the algorithm to deal with the cases below:
  - *a* is 0 (it is not a quadratic equation)
  - Δ < 0 (equation has no real roots)

Solution: new version for the algorithm

```
read a, b, c
if ??? then
    print "it is not a quadratic equation"
else
    calculate Δ
    calculate and print the roots
```

Solution: new version for the algorithm

```
read a, b, c
if a is zero then
    print “it is not a quadratic equation”
else
    calculate Δ
    calculate and print the roots
```

Solution: new version for the algorithm

```
read a, b, c
if a is zero then
    print “it is not a quadratic equation”
else
    $\Delta \leftarrow b^2 - 4ac$
    if ??? then
        print “no real roots”
    else
        calculate and print the roots
```

Solution: new version for the algorithm

```
read a, b, c
if a is zero then
    print “it is not a quadratic equation”
else
    $\Delta \leftarrow b^2 - 4ac$
    if delta is negative then
        print “no real roots”
    else
        calculate and print the roots
```

Solution: new version for the algorithm

```
read a, b, c
if a is zero then
    print “it is not a quadratic equation”
else
    $\Delta \leftarrow b^2 - 4ac$
    if delta is negative then
        print “no real roots”
    else
        x1 $\leftarrow \frac{-b+\sqrt{\Delta}}{2a}$
        x2 $\leftarrow \frac{-b-\sqrt{\Delta}}{2a}$
        print x1
        print x2
```

INF 100 – Introdução à Programação

Checking the dependencies...

```
read a, b, c
if a is zero then
    print "it is not a quadratic equation"
else
    $\Delta \leftarrow b^2 - 4ac$
    if delta is negative then
        print "no real roots"
    else
        x1 $\leftarrow \frac{-b+\sqrt{\Delta}}{2a}$
        x2 $\leftarrow \frac{-b-\sqrt{\Delta}}{2a}$
        print x1
        print x2
```

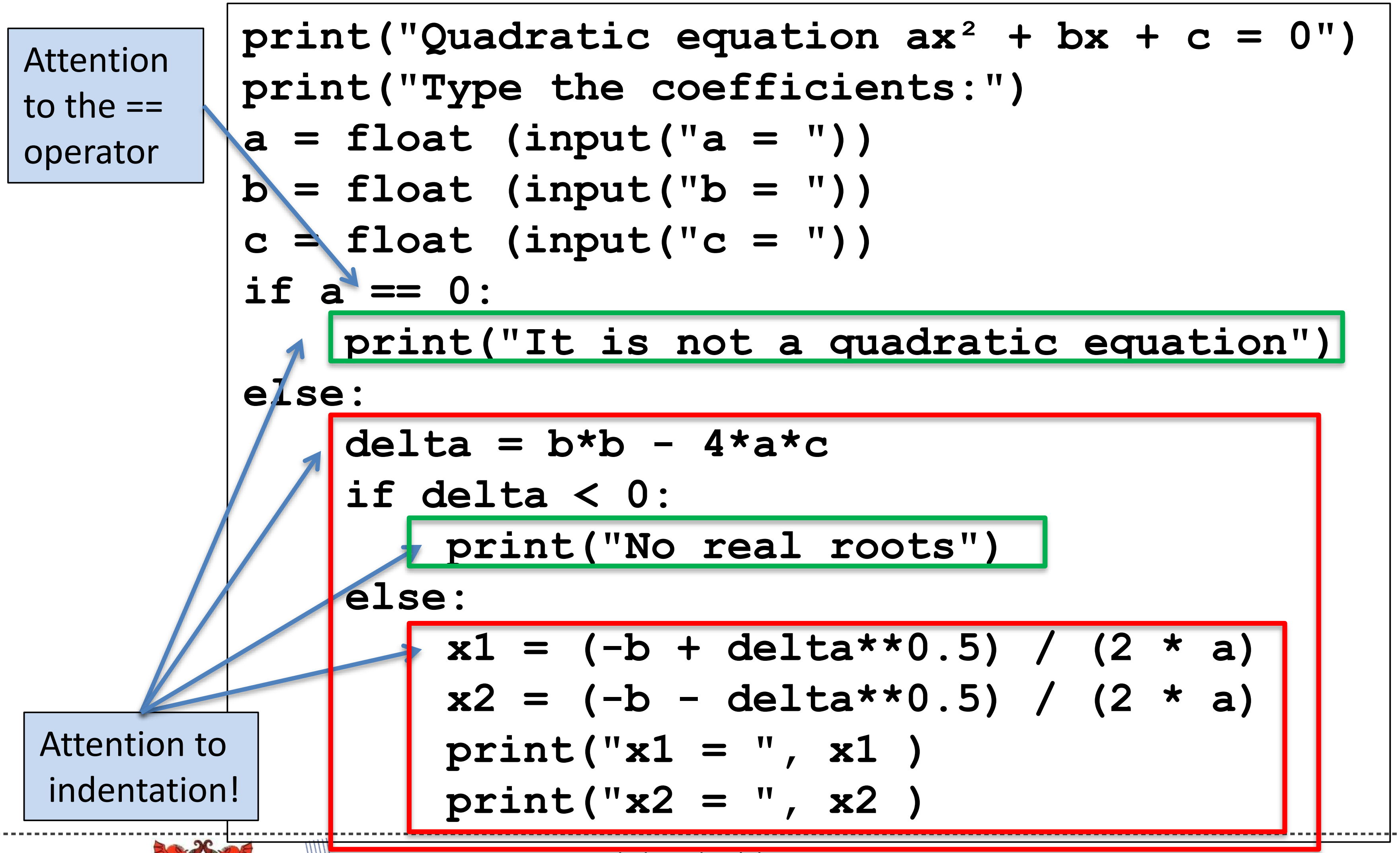
Translating to Python...

```
print("Quadratic equation ax² + bx + c = 0")
print("Type the coefficients:")
a = float (input("a = "))
b = float (input("b = "))
c = float (input("c = "))
if a == 0:
    print("It is not a quadratic equation")
else:
    delta = b*b - 4*a*c
    if delta < 0:
        print("No real roots")
    else:
        x1 = (-b + delta**0.5) / (2 * a)
        x2 = (-b - delta**0.5) / (2 * a)
        print("x1 = ", x1 )
        print("x2 = ", x2 )
```

Translating to Python...

```
print("Quadratic equation ax² + bx + c = 0")
print("Type the coefficients:")
a = float (input("a = "))
b = float (input("b = "))
c = float (input("c = "))
if a == 0:
    print("It is not a quadratic equation")
else:
    delta = b*b - 4*a*c
    if delta < 0:
        print("No real roots")
    else:
        x1 = (-b + delta**0.5) / (2 * a)
        x2 = (-b - delta**0.5) / (2 * a)
        print("x1 = ", x1 )
        print("x2 = ", x2 )
```

Translating to Python...



Attention to the == operator

```
print("Quadratic equation ax² + bx + c = 0")
print("Type the coefficients:")
a = float (input("a = "))
b = float (input("b = "))
c = float (input("c = "))
if a == 0:
    print("It is not a quadratic equation")
else:
    delta = b*b - 4*a*c
    if delta < 0:
        print("No real roots")
    else:
        x1 = (-b + delta**0.5) / (2 * a)
        x2 = (-b - delta**0.5) / (2 * a)
        print("x1 = ", x1 )
        print("x2 = ", x2 )
```

Attention to indentation!

Additional remarks

- Nested if-else commands may create situations that look ambiguous. Using comments may make the code easier to be understood.

- A common error is to use = instead of == when comparing 2 values. The Python interpreter will indicate a syntax error if you use = in an conditional expression.

Logical Operators

It is possible to write complex conditions using logical operators to combine simpler conditions.

Operator	Arity	Semantics
and	binary	returns true if both operands are true
or	binary	returns true if any of the operands are true
not	unary	returns true if the operand is false, and vice-versa

Using logical operators

To check if a variable *x* lies **inside** the range -10 to 10:

```
if -10 <= x <= 10:
...
```

OR

• • •

Using logical operators

To check if a variable *x* lies **inside** the range -10 to 10:

```
if -10 <= x <= 10:
...
```

OR

```
if x >= -10 and x <= 10:
...
```

Using logical operators

To check if a variable *x* lies **outside** the range -10 to 10:

```
if not -10 <= x <= 10:
...
```

OR

...

Using logical operators

To check if a variable *x* lies **outside** the range -10 to 10:

```
if not -10 <= x <= 10:
...
```

OR

```
if x < -10 or x > 10:
...
```

Exercise

- Write a program that reads the final grade of a student (suppose it is a value between 0 and 100), then prints a message indicating whether the student:
  - was successful (grade >= 60);
  - failed (grade < 40);
  - has the right to try the final exams (40 <= grade < 60).

Algorithm – version 1

```
read grade
if not 0 ≤ grade ≤ 100 then
    print “invalid value”
if 60 ≤ grade ≤ 100 then
    print “approved”
if 40 ≤ grade < 60 then
    print “final exam”
if 0 ≤ grade < 40 then
    print “failed”
```

Version 1 – translation to Python

```
grade = float (input("Type the grade: "))
if not 0 <= grade <= 100:
    print("Invalid value")
if 60 <= grade <= 100:
    print("Approved")
if 40 <= grade < 60:
    print("Final exam")
if 0 <= grade < 40:
    print("Failed")
```

Algorithm – version 2

```
read grade
if not $0 \leq grade \leq 100$ then
    print “invalid value”
else if grade >= 60 then
    print “approved”
else if grade >= 40 then
    print “final exam”
else
    print “failed”
```

Version 2 - translation to Python

```
grade = float (input("Type the grade: "))
if not 0 <= grade <= 100:
    print("Invalid value")
else if grade >= 60:
    print("Approved")
else if grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

else + if = elif

The **elif** clause is useful when conditional commands **if** are presented in a sequence. The **elif** clause makes the code more readable and avoids excessive indentation. Example:

```
if $cond_1$:
    $command_1$
else
    if $cond_2$:
        $command_2$
    else:
        $command_3$
```

```
if $cond_1$:
    $command_1$
elif $cond_2$:
    $command_2$
else:
    $command_3$
```

Version 2 using “elif”

```
grade = float (input("Type the grade: "))
if not 0 <= grade <= 100:
    print("Invalid value")
elif grade >= 60:
    print("Approved")
elif grade >= 40:
    print("Final exam")
else:
    print("Failed")
```

Algorithm – version 3

```
read grade
if grade > 100 then
    print “invalid value”
else if grade ≥ 60 then
    print “approved”
else if grade ≥ 40 then
    print “final exam”
else if grade ≥ 0 then
    print “failed”
else
    print “invalid value”
```

Version 3 - translation to Python

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

Algorithm – version 4

```
read grade
if 0 ≤ grade ≤ 100 then
    if grade >= 60 then
        print “approved”
    else if grade >= 40 then
        print “final exam”
    else
        print “failed”
else
    print “invalid value”
```

Version 4 - translation to Python

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

Exercise

- Extend the algorithm to check also if the student has failed because it has missed more than 25% of lectures.

Python program

```
grade = float (input("Type the grade: "))
if grade < 0 or grade > 100:
    print("Invalid value")
else:
    missing = int (input(“Missing lectures:  "))
    percentMiss = missing/30    # for INF100
    if grade < 40 or percentMiss > 0.25:
        print("Failed")
    elif grade < 60:
        print("Final exam")
    else:
        print("Approved.")
```

Logical operators revisited

- Operator **and**
- *a and b* results true if both *a* and *b* are true.

a	***b***	***a* and *b***
T	T	T
T	F	F
F	T	F
F	F	F

Logical operators revisited

- Operator **or**
- *a or b* results true if either *a* or *b* are true.

a	*b*	*a* or *b*
T	T	T
T	F	T
F	T	T
F	F	F

Logical operators revisited

- Operator **not**

a	**not** *a*
T	F
F	T

Properties of logical expressions

`not (a or b)` is equivalent to `not a and not b`

`not (a and b)` is equivalent to `not a or not b`

Example – equivalent expressions:

```
not(0 <= grade <= 100)
```

```
not(0 <= grade and grade <= 100)
```

Properties of logical expressions

`not (a or b)` is equivalent to `not a and not b`

`not (a and b)` is equivalent to `not a or not b`

Example – equivalent expressions:

```
not(0 <= grade <= 100)
```

```
not(0 <= grade and grade <= 100)
```

```
not(0 <= grade) or not(grade <= 100)
```

Properties of logical expressions

`not (a or b)` is equivalent to `not a and not b`

`not (a and b)` is equivalent to `not a or not b`

Example – equivalent expressions:

```
not(0 <= grade <= 100)
```

```
not(0 <= grade and grade <= 100)
```

```
not(0 <= grade) or not(grade <= 100)
```

```
grade < 0 or grade > 100
```

Logical values

- The value of a logical test (True or False) may be stored in a variable:

```
x = 1
y = 2
a = y < 5
b = x == y
c = (x < y) and a
print("2 < 5:", a )
print("x == y:", b )
print("(x < y) and a:", c )
```

Logical values

- The value of a logical test (True or False) may be stored in a variable:

```
x = 1
y = 2
a = y < 5
b = x == y
c = (x < y) and a
print("y < 5:", a )
print("x == y:", b )
print("(x < y) and a:", c )
```

```
y < 5: True
x == y: False
(x < y) and a: True
```

Operators – table of precedence

Priority	Operator	Example
1	**	x ** 3
2	- (unary)	-x
3	* / // %	x / y
4	+ -	x - y
5	< <= > >= == !=	x < y
6	not	
7	and	
8	or	

Examples

Expression	Result
2 >= 1 or 5 != 4	
not 2 < 4	
4 == 2 + 2 and 3 > 8	
8 % 2 == 4 or 1.6 <= 3.0 / 2	
not 5 > 9 or 1 + 2 == 3 and 2 <= 7	

Examples

Expression	Result
2 >= 1 or 5 != 4	True
not 2 < 4	False
4 == 2 + 2 and 3 > 8	False
8 % 2 == 4 or 1.6 <= 3.0 / 2	False
not 5 > 9 or 1 + 2 == 3 and 2 <= 7	True

Exercise

- Write a program that reads the date of birth of a person and a second date, then calculates and prints the age of the person in the second date. Sample execution:

```
Date of birth:
Type day: 1
Type month of birth: 3
Type year of birth: 1990
Second date:
Type day: 1
Type month: 10
Type year: 2000
The age is 10
```

Exercise

- Another sample execution:

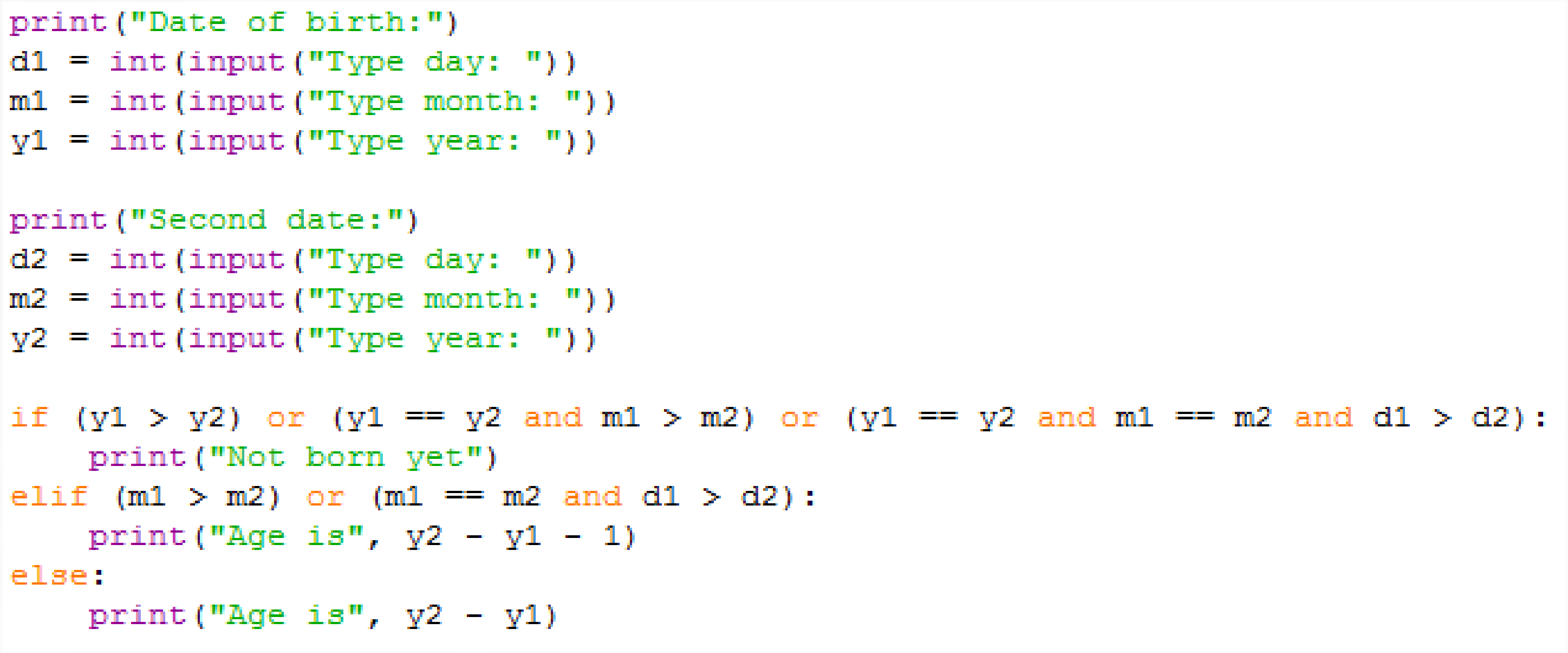
```
Date of birth:
Type day: 1
Type month of birth: 3
Type year of birth: 1990
Second date:
Type day: 10
Type month: 2
Type year: 1995
Age is 4
```

Exercise

- Another sample execution:

```
Date of birth:
Type day: 2
Type month of birth: 4
Type year of birth: 1988
Second date:
Type day: 3
Type month: 3
Type year: 1988
Not born yet
```

Solution



```python
print("Date of birth:")
d1 = int(input("Type day: "))
m1 = int(input("Type month: "))
y1 = int(input("Type year: "))

print("Second date:")
d2 = int(input("Type day: "))
m2 = int(input("Type month: "))
y2 = int(input("Type year: "))

if (y1 > y2) or (y1 == y2 and m1 > m2) or (y1 == y2 and m1 == m2 and d1 > d2):
    print("Not born yet")
elif (m1 > m2) or (m1 == m2 and d1 > d2):
    print("Age is", y2 - y1 - 1)
else:
    print("Age is", y2 - y1)
```

Exercise

- Suppose that x, y and z are real values, and $\boldsymbol{a}$ is the average of these values.

- We want to calculate n, the number of values which are greater than $\boldsymbol{a}$.

...what values can $\boldsymbol{n}$ assume?

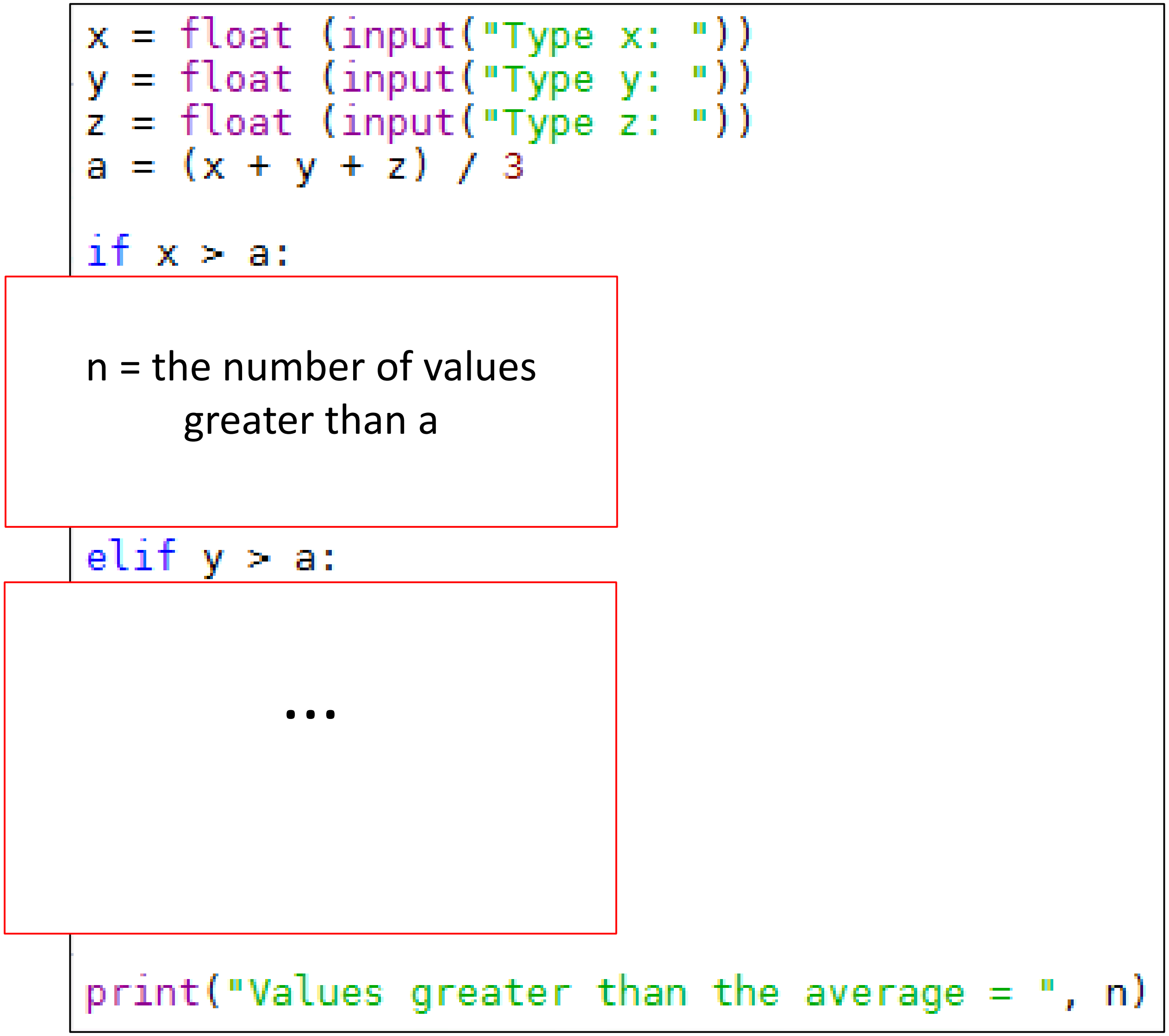
Exercise

- Suppose that x, y and z are real values, and a is the average of these values.

- We want to calculate n, the number of values which are greater than a (n must be 0, 1 or 2, because it is impossible to have 3 values greater than the average).

- Example: Suppose x=5, y=7, z=18. Then the average a is 10. In this case, only z is greater than a, therefore we have n=1.

Solution



```
x = float (input("Type x: "))
y = float (input("Type y: "))
z = float (input("Type z: "))
a = (x + y + z) / 3

if x > a:
```

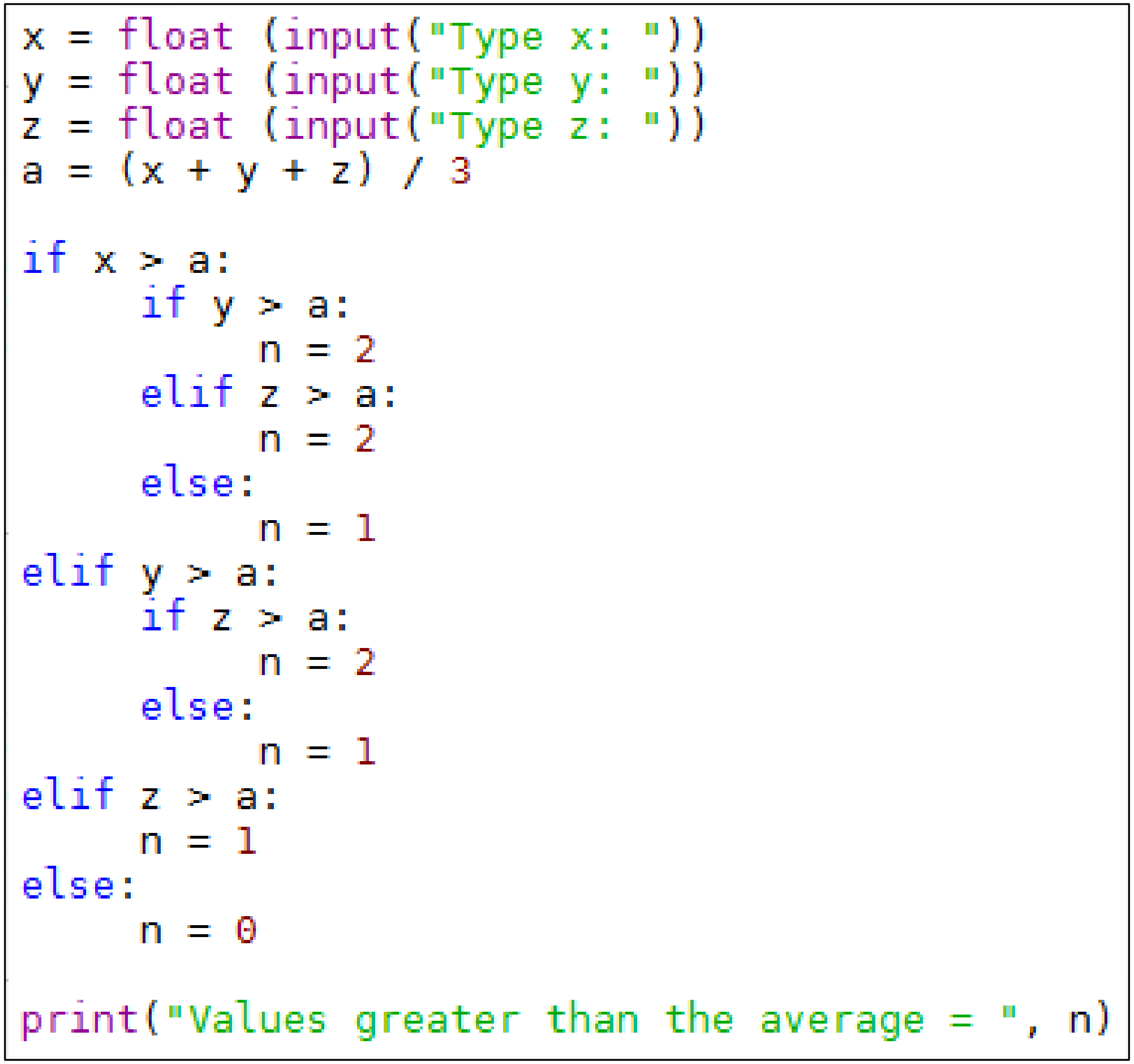
n = the number of values greater than a

```
elif y > a:
```

...

```
print("Values greater than the average = ", n)
```

Solution



```
x = float (input("Type x: "))
y = float (input("Type y: "))
z = float (input("Type z: "))
a = (x + y + z) / 3

if x > a:
    if y > a:
        n = 2
    elif z > a:
        n = 2
    else:
        n = 1
elif y > a:
    if z > a:
        n = 2
    else:
        n = 1
elif z > a:
    n = 1
else:
    n = 0

print("Values greater than the average = ", n)
```

Exercise

- Complete the 3 conditions in the code below, in a way that it works in the same way that the previous solution.

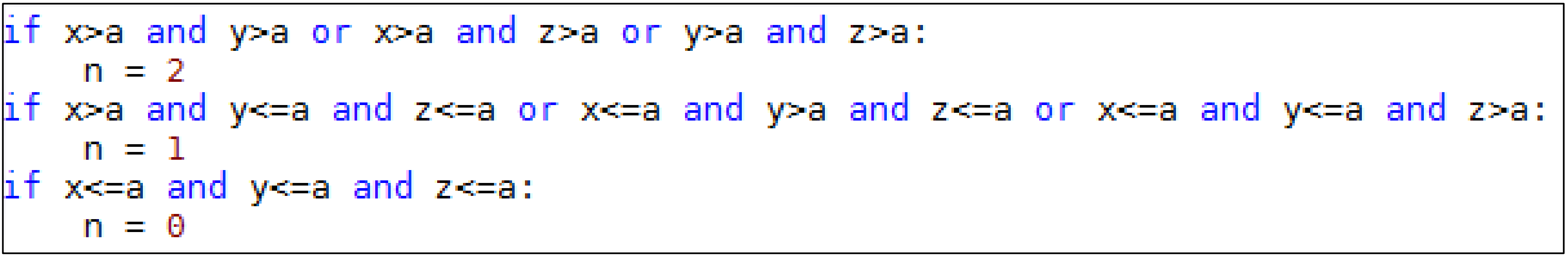
```
if            ???             :
    n = 2

if            ???             :
    n = 1

if            ???             :
    n = 0
```

Solution for the exercise

- Complete the 3 conditions in the code below, in a way that it works in the same way that the previous solution.



```
if x>a and y>a or x>a and z>a or y>a and z>a:
    n = 2
if x>a and y<=a and z<=a or x<=a and y>a and z<=a or x<=a and y<=a and z>a:
    n = 1
if x<=a and y<=a and z<=a:
    n = 0
```

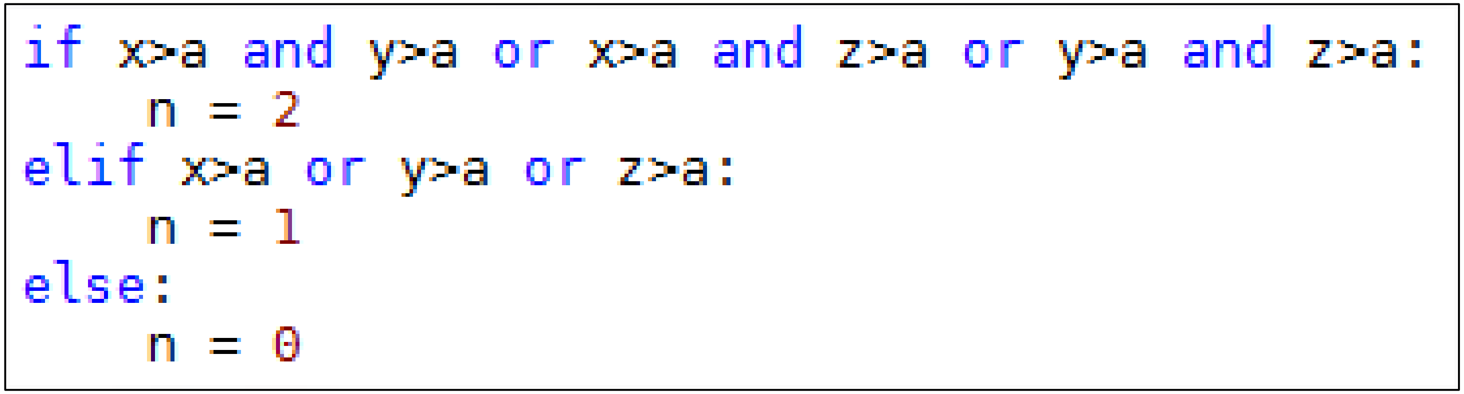
Exercise

- Now complete the 3 conditions in the code below, in a way that it works in the same way that the previous solution.

```
if                ???                :
   n = 2
elif                ???                :
   n = 1
else:
   n = 0
```

Solution for the exercise

- Now complete the 3 conditions in the code below, in a way that it works in the same way that the previous solution.



```
if x>a and y>a or x>a and z>a or y>a and z>a:
    n = 2
elif x>a or y>a or z>a:
    n = 1
else:
    n = 0
```

Another solution for the same problem

- We may find out how many values are greater than the average using another technique:
  - initialize variable n with zero;
  - sequentially compare a with x, y and z;
  - whenever a is greater than a value, increment variable n by one unit.

Solution in Python...

```
x = float(input("Type x: "))
y = float(input("Type y: "))
z = float(input("Type z: "))
a = (x + y + z) / 3

n = 0
if a > x:
    n += 1
if a > y:
    n += 1
if a > z:
    n += 1

print("Values greater than the average =", n)
```

Homework

Do the exercises B.2, B.3 and B.8 defined in *run.codes*.