

4: Conditional statements

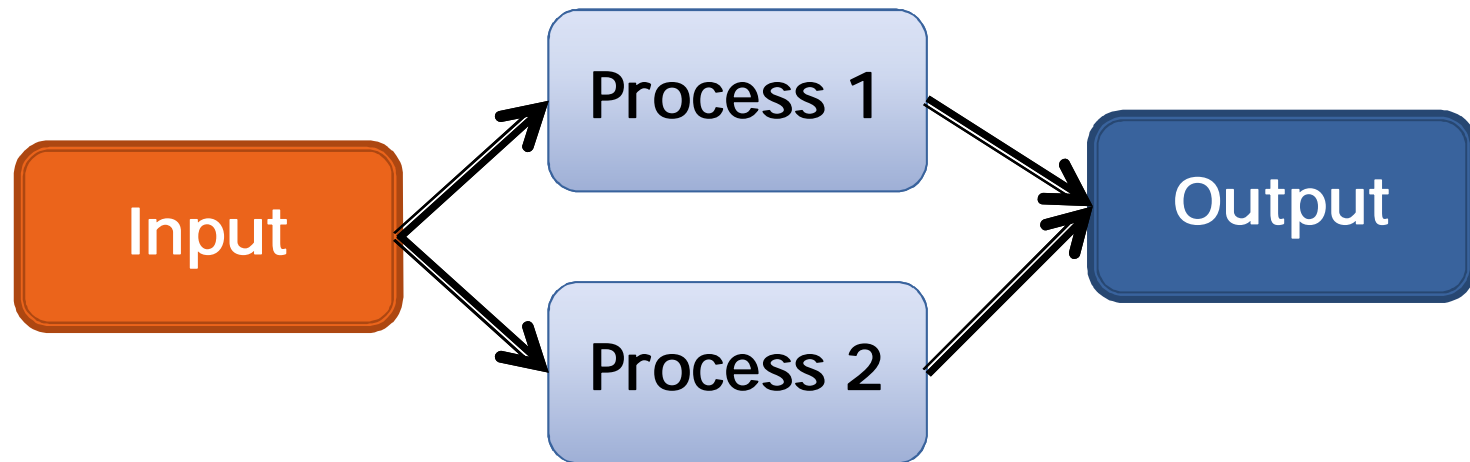
The execution sequence

- } So far all of our programs have been following the same sequential path:



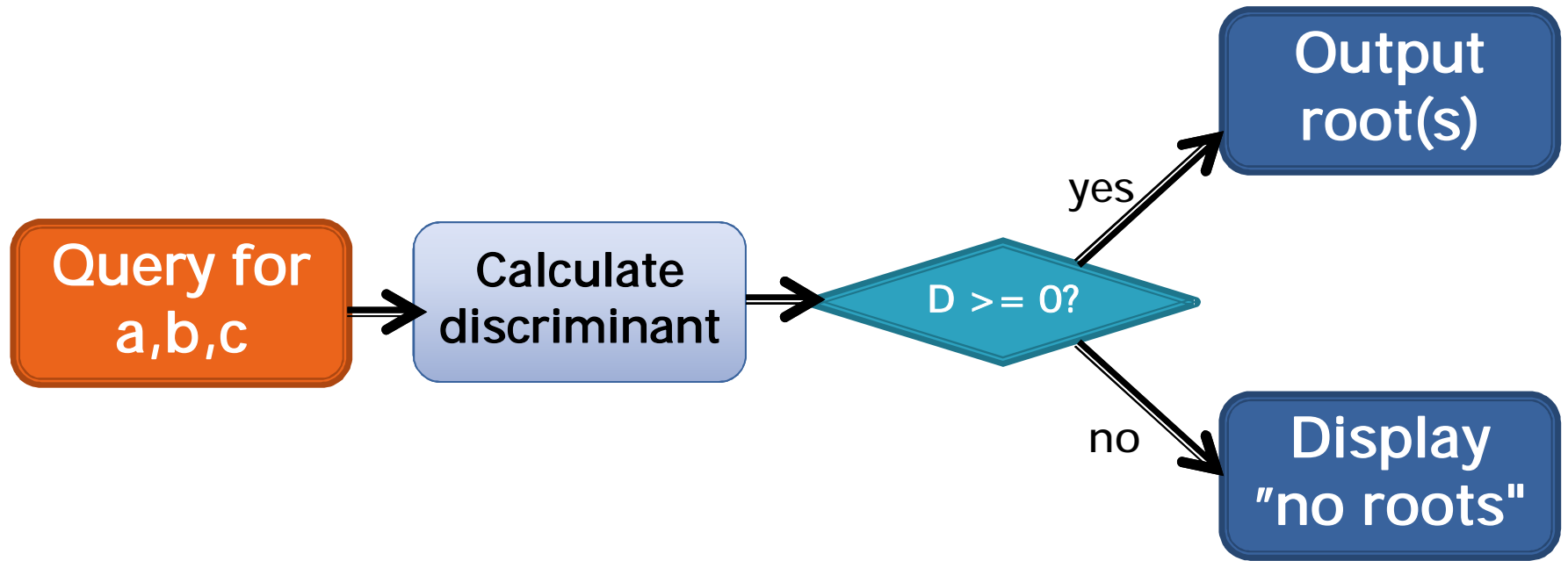
Conditional execution

} ...however, programs usually need to change their performance according to some conditions:



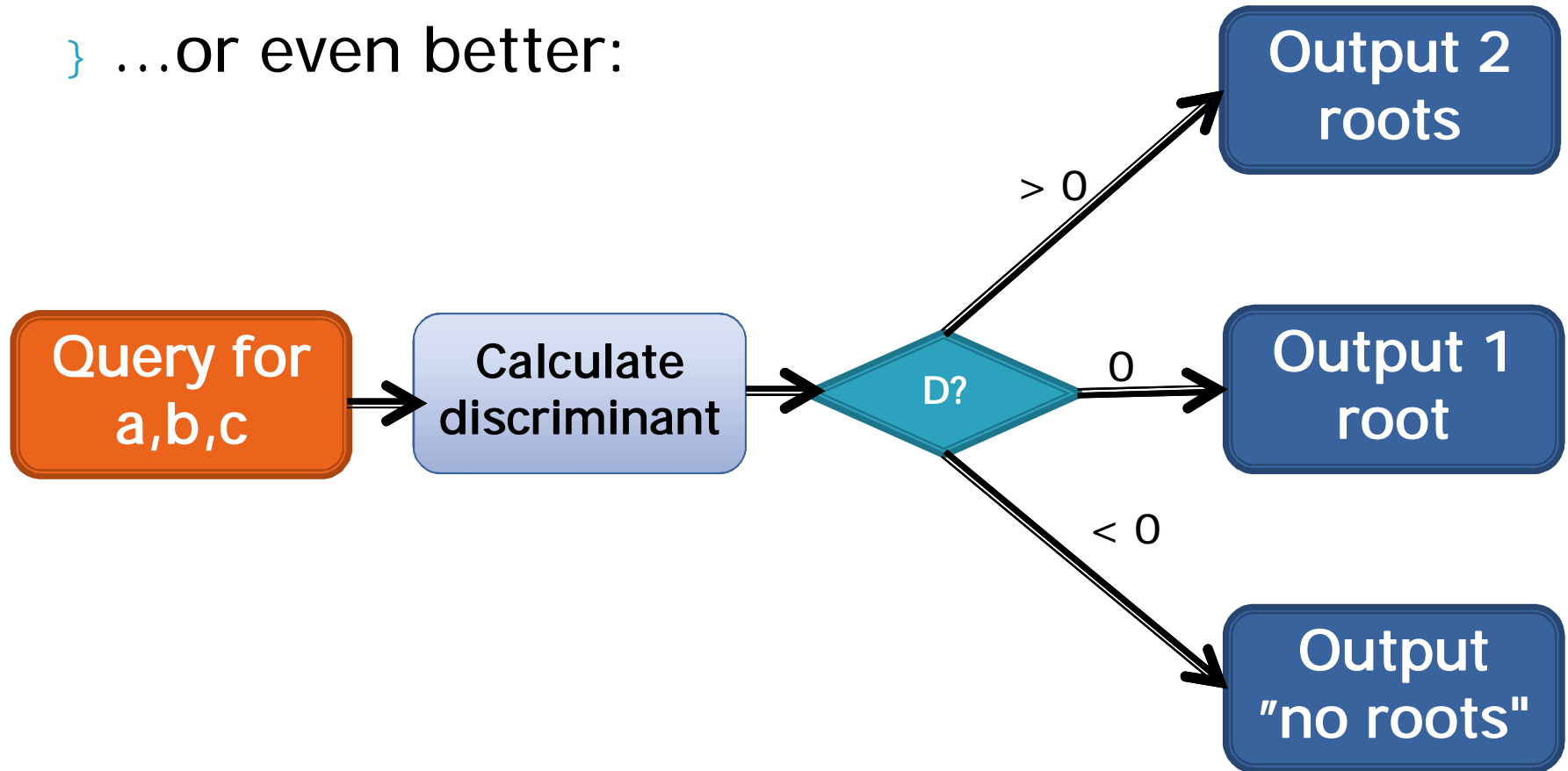
For example

} Assignment 1, Q6:




For example

} ...or even better:



Conditional execution (2)


- } Execute a program block only if given condition is True.
 - } Also possible to execute another block (or blocks), if condition is False.
 - } For example: process the two-word sentence only if the input string contains exactly one space.
- 

if statement in Python


- } The conditional execution in Python can be implemented by using the **if** statement.
- } The syntax of the **if** statement in Python is:

```
if condition:  
    statement block
```

If statement in Python


- } The statements in the block following the **if** statement are executed only, if the condition evaluates to **True**.
 - } If the condition evaluates to **False**, the execution continues at the first line after the block.
 - } Note, that the lines in the statement block are indented with a tab or with same number of spaces
- 

Program blocks

- } All consecutive code lines with the same or more indentation belong to the same block
 - } Blocks can contain other blocks (which can contain other blocks and so on...)
 - } In Python the block consists of one or more statements. It is not possible to create an empty block.
- 

Example

```
a = input ( "Give a number:" )
if a > 0:
    print a
    a = a * 2
    if a < 10:
        a = -a
        print a ** 2
    print "Getting close to end"
    b = a * 2
print "Bye!"
```



Example

```
a = input ("Give a number: ")
if a > 0:
    print a
    a = a * 2
    if a < 10:
        a = -a
        print a ** 2
    print "Getting close to end"
    b = a * 2
print "Bye!"
```

Block 1

Example

```
a = input ("Give a number: ")
```

Block 1

```
if a > 0:
```

```
    print a
```

Block 2

```
    a = a * 2
```

```
    if a < 10:
```

```
        a = -a
```

```
        print a ** 2
```

```
    print "Getting close to end"
```

```
    b = a * 2
```

```
print "Bye!"
```

Example

```
a = input ("Give a number:")  
if a > 0:  
    print a  
    a = a * 2  
    if a < 10:  
        a = -a  
        print a ** 2  
    print "Getting close to end"  
    b = a * 2  
print "Bye!"
```

Block 1

Block 2

Block 3

Conditions and boolean variables

- } The simplest way to express a condition is to use a variable of type **boolean**.
- } Boolean variables have only two possible values, *True* or *False*:

```
myVariable = True  
outputNumbers = False
```

Examples

- } Output the value assigned in variable myName only if the value of boolean variable outputValue is true:

```
if outputValue:  
    print myName
```


Examples (2)

} What do the following programs output?

```
var = True
if var:
    print "First"
print "Second"
```

```
var = False
if var:
    print "First"
print "Second"
```

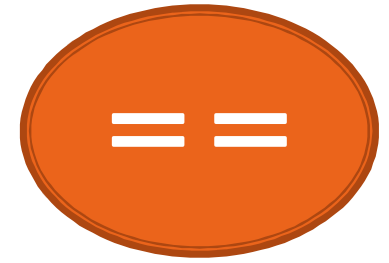

Comparison operators

- } Usually we need to compare values of two or more **operands** (i.e. variables, constant values etc.)
 - } Hence, Python has a number of **comparison operators**.
 - } The expression with comparison operators evaluates into **True** or **False**.
- 

Some comparison operators in Python

Operator	Description
==	Equal to
>	Greater than
<	Smaller than
>=	Greater than or equal to
<=	Smaller than or equal to
!=	Not equal to

1. Equality



- } The equality is checked by the == operator
- } Note, the difference to assignment operator
- } The statement

```
expression1 == expression2
```

evaluates to True, if the value of expression1 is exactly the same than the value of expression2.

Examples

} Conditions that evaluate to True:

`3 == 3`

`11 == 2 + 4 + 5`

`"hello" == "hel" + "lo"`

} Conditions that evaluate to False:

`4 == 5`

`a = 3`

`a == 4`

`True == False`

`2.5 == 5 / 2`




Comparing boolean variables

} Note, that although syntactically this will work...

```
if value == True:  
    print "It's true"
```

} ...it's easier (and better) to write it like this:

```
if value:  
    print "It's true"
```



Comparing boolean variables

} Because the condition here evaluates to True...

```
if value == True:  
    print "It's true"
```

} ...the code above is equivalent to code below:

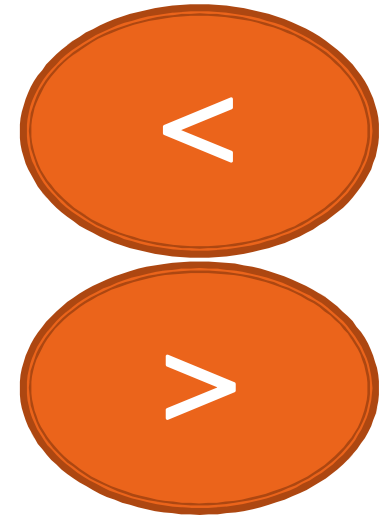
```
if value:  
    print "It's true"
```

Comparing boolean.. (3)

- } Interestingly, boolean variables have matching integer values (True == 1, False == 0)

```
print True == 1 # outputs True  
print False + 2 #outputs 2
```

Smaller / greater than



- } Comparison for values smaller or greater than like in mathematics
- } Works with numbers AND with strings.
- } The comparison of strings is done by their alphabetical order; note, that "a" ≠ "A"!

Examples

} Conditions that evaluate to True:

`5 > 3`

`4 < 3 + 2`

`"a" < "z"`

} Conditions that evaluate to False:

`4 + 2 < 5`

`"car" < "alpha"`



Smaller / greater than or equal to

} Like mathematical operators \leq and \geq

} Example:

```
a = 4
```

```
b = 6
```

```
print a <= b
```

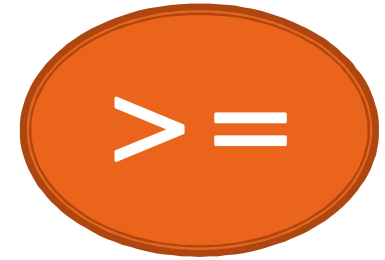
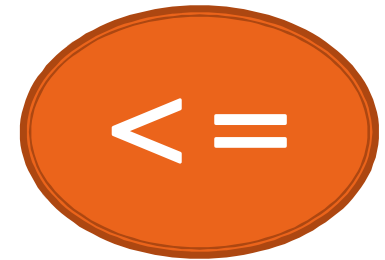
à **True**

```
print a + 2 <= b
```

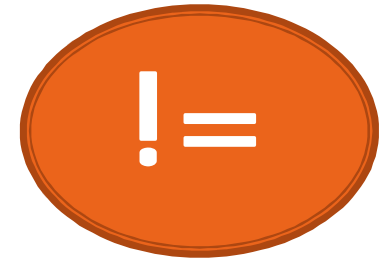
à **True**

```
print a >= b
```

à **False**



Not equal to



} To check whether the values are NOT equal, we can use the != operator

} The statement

```
expression1 != expression2
```

returns True, if expression1 is not exactly equal to expression2

Examples

} Conditions that evaluate to True:

`3 != 5`

`"aaa" != "aab"`

`(3 > 2) != False`

} Conditions that evaluate to False:

`4 != 2 + 2`

`"abcabc" != "abc" * 2`

`(3 == 1 + 2) != True`

Equality and inequality

- } Note, that testing for equality for True is always equivalent to testing for inequality to False.

`if a == True:`  `if a != False:`

Evaluating boolean statements

- } As every expression containing boolean statements evaluates to True or False, this

```
if myValue == True:
```

- } ..is equivalent to

```
if myValue:
```



Evaluating boolean statements (2)

} And this

```
if myValue == False:
```

} ..is equivalent to

```
if not myValue:
```



Example1

- } A program that queries the user for a number, and outputs "Is odd" if the number is an odd number:

```
number = input("Give a number: ")  
if number % 2 == 1:  
    print "Is odd!"
```


Example 2

} A program that queries the user for two numbers and outputs the larger:

```
a = input("Give 1st number :")
b = input("Give 2nd number :")
if a > b:
    print a
if b > a:
    print b
```

The else statement

} However, it is often necessary to "do something, if condition is true, and something else, if it's not".

} This could be achieved with two if statements...

```
if condition == True:
```

```
    ...do something here...
```

```
if condition == False:
```

```
    ...do something else here..
```



The else statement (cont.)

- } ...but, since the state of the condition can change in the first block (and for convenience reasons) we should use the **else** statement. Syntax:

```
if condition:  
    block1  
else:  
    block2
```

- } Now, block1 is executed if the condition is True, and block2 if the condition is False.

Example 2.2

- } A rewrite of the program that queries the user for two numbers and outputs the larger:

```
a = input("Give 1st number :")
b = input("Give 2nd number :")
if a > b:
    print a
else:
    print b
```

Note the difference!

} Are the programs equivalent?

```
a = input("Give 1st number :")
b = input("Give 2nd number :")
if a > b:
    print a
else:
    print b
```

```
a = input("Give 1st number :")
b = input("Give 2nd number :")
if a > b:
    print a
if b > a:
    print b
```

Example 3

- } Query the user for a value, and display a message and the value depending whether the value is an odd number or not:

```
a = input("Give a number :")
if a % 2 == 0:
    print "The value was: ", a
    print "...not an odd number"
else:
    print "The value was:", a
    print "...which is an odd number!"
```

Joining conditions

- } Multiple conditions can be joined into one conditional statement by using Python's logical operators.
- } Note the similarities to binary operators, seen in CS-course

Logical operators in Python

Operator	Description
and	The result is true, if both operands are True
or	The result is True, if atleast one of the operands is True
not	Logical not à complements the value of an expression

- Note, that there is no logical XOR defined in Python.

Logical operators in Python (2)

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Examples

} Conditions that evaluate into True:

```
3 > 2 and 3 < 5
```

```
myStr = "abc"
```

```
myStr < "bcd" and len(myStr) == 3
```

```
True or (2 > 3)
```

```
9 >= 11 or "abc" == "ab" + "c"
```

```
not (5 < 4)
```

Example

- } Query user for an email address, and check if it is valid (i.e. contains a "@" character and is at least 5 characters long)


```
email = raw_input("Give email :")
if len(email) >= 5 and email.find("@") > -1:
    print "This could be a valid address"
else:
    print "This is not a valid address."
```

Example 2

- } Query user for a 4 to 10 character string, and display an error, if it's longer or shorter:

```
myStr = raw_input("Give 4-10 char. string: ")
if len(myStr) < 4 or len(myStr) > 10:
    print "String is too short or too long!"
else:
    print "Thank you!" # continue to process...
```

Nested if statements

- } The blocks following the if and else statements can contain any statements (and any number of statements), including *other if statements*.
 - } In some cases, the nested if structures could be replaced by using joined conditions. In other cases they can't. J This is just something you need to consider in each situation...
- 

Example

- } Query the user for a number, and if it is positive, display whether it is an odd number or not:

```
number = input("Give me a number :")
if number > 0:
    print "The number is positive..."
    if number % 2 == 0:
        print "...and an even number"
    else:
        print "...and an odd number"
else:
    print "The number is negative."
```

The elif statement

- } The **elif** (as in **else if**) statement in Python can be used to test if condition is true after another condition was false. The syntax is

```
if condition:
    statement block 1
elif condition2:
    # condition1 == False, condition2 == True
    statement block 2
else:
    # condition 1 and condition2 == False
    statement block 3
```

- } Note, that the final else block is optional. There can be any number of elif statements after the if block.

Example

- } Query user for number and display whether it's positive, negative or zero:

```
number = input("Give a number :")  
if number > 0:  
    print "positive"  
elif number < 0:  
    print "negative"  
else:  
    print "zero"
```


Example (2)

} Query for two numbers and display larger

```
a = input("Give 1st number :")
b = input("Give 2nd number :")
if a > b:
    print a
elif b > a:
    print b
else:
    print "The numbers are equal."
```

