HKU

ECTTP: Conditions

Valentijn Muijrers

https://github.com/vmuijrers/ECTTP

Course Overview

- Week One: Variables
- Week Two: Operators
- **Week Three: Conditions ←**
- Week Four: Loops
- Week Five: Functions
- Week Six: Tuples
- Week Seven: **First Test**
- Week Eight: Lists

- Week Eleven: Classes and Objects
- Week Twelve: Classes and Objects
- Week Thirteen: Classes and Objects
- Week Fourteen: Classes and Objects
- **Second Test!**

Our Super Powers so far…

- Variables! (Int, String, Boolean and Float)
- They can have any name!
- And you can give them values with the '=' operator
- string_mySuperPowerVariable = "Awesome!"

...More Super Powers!

- Operators!
- Variables can be changed and manipulated with operators! (+, -, *, /)
- Always be sure the input values are of the same type!
- int_myInteger = 3 **+** 5 <<< 8
- int_myInteger = 3 **+** "Kipje" <<< Error!

- There are exceptions:
- string_myString = "Kipje" ***** 3 <<< "KipjeKipjeKipje"

Boolean Expressions

- Boolean types only have two values: True or False
- boolean_myBoolean = True #Set the variable to True
- If (boolean_myBoolean **==** True):#This is an if-statement
  - #Do something here
- Boolean expressions ask a question and produce a yes or no result we use to control how a program flows

Comparison Operators

- Boolean Expressions use comparison operators to evaluate if something is true or false, yes or no

- Some examples of comparison operators are:
- == (compares two values, True if the values are **equal**)
- > (compares two values, True if the left side is **larger**)
- < (compares two values, True if the left side is **smaller**)
- >= (compares two values, True if the left side is **equal** or **larger**)
- <= (compares two values, True if the left side is **equal** or **smaller**)
- != (compares two values, True if the values are **not equal**)

- Comparison operators look at variables but **do not change** variables

Conditional Statements

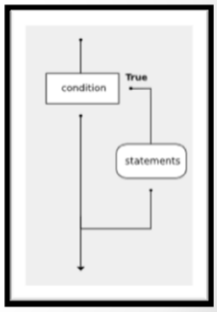
- The Comparison operators can be use to check conditional statements
- This can be used to change the behavior of the program accordingly
- An example is the if-statement:

- if (<Condition>):
  - #If the condition is True then this code executes
- else:
  - #If the condition is False then this code executes

Example

```
x = 10
if ( x == 10):
        x = x + 1
#because x is 10 the operator == resolves to True and the code will be executed

y = "Hello"
If ( y == "hello" ):
        y = "CAPS"
# This code does not get executed because "Hello" is not equal to "hello"
```



Indentation

- Indentation matters in Python.
- If you have a statement that
- ends in a **:** you can add lines
- of code underneath it tabbed
- in with one tab and those lines
- of code are grouped within the
- **scope** of that condition

- When you un-indent you leave
- this **scope** and are back
- *outside* the conditional block

Indentation Example

```
x = 5
y = 7
if ( x == 5 or y == 6 ):
        x = 10
        y = 10
        print ("This is indented!")
print("This is not indented")
```

```
This is not indented.
  This is indented.
This is not indented.
```

Scope

- The Scope determines the life-time of a variable in memory

```
y = 10
if ( y == 10 ):
        x = "I am alive!"
        #The indent creates a scope, at the end of the
        scope, the variable x will die
print x
#this will give an error in most languages because the
variable is no longer in memory (in processing this will still
work, but it is bad practise!)
```

Logical Operators

- It is possible to chain multiple conditions together using Logical Operators
- The following are defined:

- **AND** #is True if both the condition on the right is True and the condition on the left is True, otherwise False
- **OR** # is True if either left or right or both are True, otherwise False
- **NOT** #Flips the value to the opposite, so True becomes False, False becomes True

Example AND

```
#The And operator
x = 6
y = "Kaasje"

if x == 6 and y == "Kaasje":
        print "both conditions are true"
        #Both sides of the and-operator are True, so this
        statement is executed
```

Example OR

```
#The Or operator
x = 6
y = "Kaasje!!!"

if x == 6 or y == "Kaasje":
        print "one condition is true"
        #One of the sides of the operator is True, so this
        statement will be executed

#Note that only if BOTH left and right side are False,
the OR-operator will resolve to False
```

Example NOT

```
x = False

if not x:
        y = 5
        #The NOT operator flips the value of x, so that
        the condition becomes True and the if-
        statement is executed

#sometimes you will see '!' instead of not
```

Can You Follow Instructions?

- Rule 1: Execute the rules once when Valentijn says "Go!"
- Rule 2: **IF** you sit next to a girl, raise your hands
- Rule 3: **IF** you have **not** raised your hands, stand up **ELSE** applaud
- Rule 4: **IF** you are standing up, bow **ELSE** stand up
- Rule 5: **IF** you are bowing **and not** applauding, sit down **ELSE** applaud

Order of Execution

Python will always evaluate the arithmetic operators first (** is highest, then multiplication/division, then addition/subtraction). Next comes the relational operators.
Finally, the logical operators are done last.

Level Category Operators:

- 8 parentheses (,)
- 7(high) exponent **
- 6 multiplication *,/,//,%
- 5 addition +,-
- 4 relational ==,!=,<=,>=,>,<
- 3 logical not
- 2 logical and
- 1(low) logical or

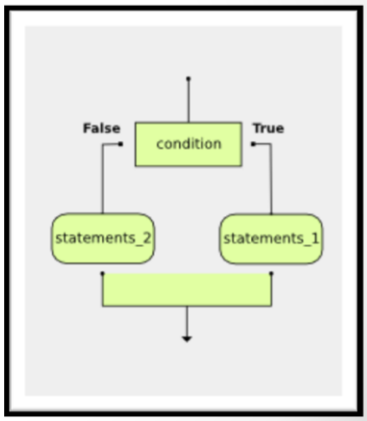
What if...else

If-statements can have second parts!

```
x = 11

if x == 10:
        STATEMENTS_1
else:
        STATEMENTS_2
```

if statement one is true run the first set of statements, else run the statements in the second block



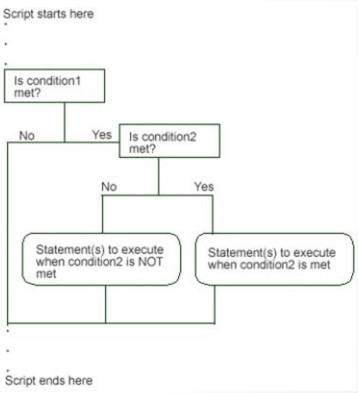
If-ception

You can **nest** if-statements inside each other:

```
if x < y:
        print("x is less than y")
else:
        if x > y:
                print("x is greater
                than y")
        else:
                print("x and y must
                be equal")
```

Flow

- So with if-statements, you can make choices in your program, it determines the **Flow** of the program



Chained Conditional

- Python provides an alternative way to write nested selection such as the one shown in the previous section. This is sometimes referred to as a **chained conditional**

```
if x < y:
        print("x is less than y")
elif x > y:
        print("x greater than than y")
elif x != y:
        print("x is not equal to y")
else:
        print("x and y are equal")

#The elif can be used multiple times to create a long chain of conditionals
```

Codecademy

- Codecademy is a great way to learn how to program! It has instructions and small exercises to become a pro coder!

- Now let's practise!
- https://www.codecademy.com/learn/python

CodingBat

- Now let's practise some more:
- http://codingbat.com/python

Third lab is online

https://github.com/vmuijrers/ECTTP/blob/master/Labs/Lab_3.md

#For examples/tutorials and references!
py.processing.org

#For more practise with python!
codecademy.com