Python workshop

Week 2: Make choices and reuse code

barbera@van-schaik.org

Overview of this workshop series

- Week 1: Writing your first program
- Week 2: Make choices and reuse code
- Week 3: Loops and strings
- Week 4: Files and lists
- Week 5: Dictionaries and tuples

Acknowledgments: Structure of the workshop follows the book
“Python for informatics” by Charles Severance.
Several examples are from this book or the accompanying slides.

Guido van Rossum

- Master's degree mathematics and computer science at University of Amsterdam
- Worked for several big IT companies
- Invented Python
- “Computer programming for Everybody”

https://en.wikipedia.org/wiki/Guido_van_Rossum

The name “Python”

https://youtu.be/QgaRd4d8hOY

Open source

- Software
- Electronics
- Digital content
- Food and drinks
- Medicine
- Science
- Fashion

- GPL, Apache, MIT, etc
- Creative Commons

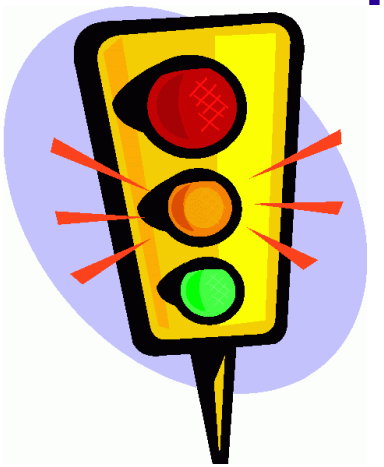
https://en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses
https://en.wikipedia.org/wiki/Creative_Commons_license

This workshop

- Content a combination of mine and that of Charles Severance (Creative Commons, attribution)
- Code: Idem. My code under copy-left (no restriction)
- Images: Not mine. Sources will be added to the notes a.s.a.p. Please contact me if images need to be removed or acknowledged. If you use these slides, be aware of this and replace them

Making choices

Python and indentation



```
if traffic_light == “green”:
    print “Go! Go! Go!”
else:
    print “Stop!”
```

Change settings/preferences of your editor
Tabs to spaces = True
Maybe you need to check a box somewhere

Sublime editor

```
// Settings in here override those in
"Default/Preferences.sublime-settings",

// and are overridden in turn by file type specific settings.

{

    "tab_size": 4,

    "translate_tabs_to_spaces": true

}
```

Datatype: boolean


```
>>> 1 + 2 == 3
>>> 1 + 1 == 3
```

Comparison operators

```
x != y              # x is not equal to y
x > y               # x is greater than y
x < y               # x is less than y
x >= y              # x is greater than or equal to y
x <= y              # x is less than or equal to y
x is y              # x is the same as y
x is not y          # x is not the same as y
```

Logical operators

```
x > 0 and x < 10   # both need to be True
```

```
n % 2 == 0 or n % 3 == 0
```

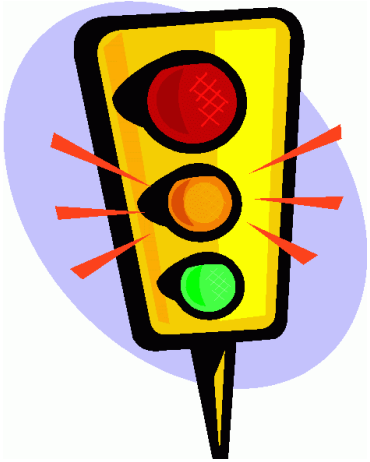
Do you want coffee or tea? Answer: yes

```
not (x > y)    # negates the expression
```

```
>>> 17 and True
```

Python considers 0 as False and all other numbers as True

Making choices: if else



```
if traffic_light == “green”:
    print “Go! Go! Go!”
else:
    print “Stop!”
```

```
if x > 0:
    print “x is positive”
else:
    print “x is not positive”
```

Be aware that it is not necessarily negative

Question: what will happen here?

```
x = 2
if x > 2:
    print “Code in if block is executed”
    print “and this line too
print “Finished”
```

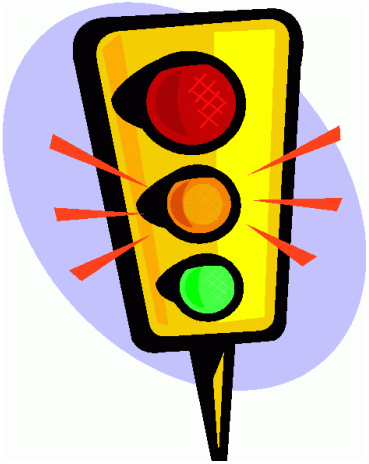
This is False

Change it to: x >=2

Both lines in the indented block are executed if statement is True or skipped if statement is False

This is not part of the if-block and in this case always executed

Making choices: if elif else



```
if traffic_light == “green”:
    print “Go! Go! Go!”
elif traffic_light == “orange”:
    print “You should stop.. if possible”
else:
    print “Stop!”
```

Order of execution

```
x = 25
if x < 10:
    print “Number is below ten”
elif x < 20:
    print “Number is below twenty”
elif x < 40:
    print “Number is below forty”
elif x < 30:
    print “Number is below thirty”
else:
    print “Number is forty or higher”
```

Nested conditionals

```
answer = raw_input("Do you want something to 'drink' or 'eat'? ")

if answer == "drink":
    drink = raw_input("Do you want 'coffee' or 'tea'? ")
    if drink == "coffee":
        print "There you go. Coffee for you."
    elif drink == "tea":
        print "Tea for you it is."
    elif drink == "yes":
        print "Ha ha, funny... Not"
    else:
        print "Sorry, we don't have", drink
else:
    print "Ok, I'll fix you something to eat"
```

coffee-or-tea.py on the wiki

Catch exceptions

```
inp = raw_input('Enter Fahrenheit Temperature:')

fahr = float(inp)

cel = (fahr - 32.0) * 5.0 / 9.0

print cel
```

Run program, and enter: 72
Run it again and try: blah

Catch exceptions

```
inp = raw_input('Enter Fahrenheit Temperature:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print cel
except:
    print 'Please enter a number'
```

Run program, and enter: 72
Run it again and try: blah

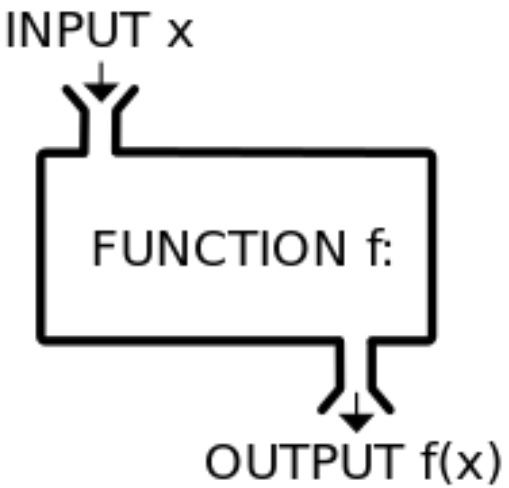
Reuse code

Functions

- raw_input(“What is the answer to everything?”)
- type(42)
- int(45.6987)

Function name

Input / Argument



A function “takes” an argument and “returns” a result

Built-in functions

- Type conversion: str(13), int('13'), float(13)

```
>>> len(“Once upon a time”)
>>> min(“Once upon a time”)
>>> max(“Once upon a time”)
```

Using more functions

- If a function is not already built-in you can import it

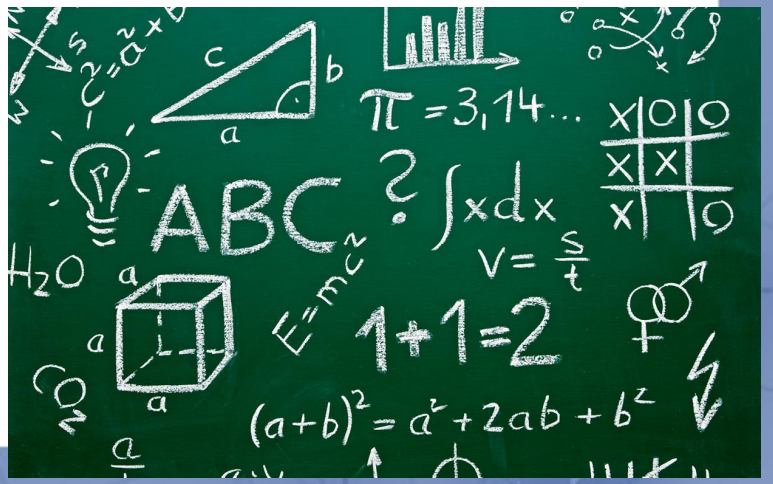
```
import random

for i in range(10)
    x = random.random()
    print x
```

The “random” library

```
>>> import random

>>> random.randint(5, 10)

>>> t = [“a”, “b”, “c”]
>>> random.choice(t)
```

The “math” library

```
import math


degrees = 45
radians = degrees / 360.0 * 2 * math.pi
sinus = math.sin(radians)


print “degrees”, degrees
print “radians”, radians
print “sinus”, sinus
```



Create your own functions

```
def printTekst ():
    print "Ik heb een potje met vet,"
    print "Al op de tafel gezet."
    print "Ik heb een potje potje potje potje ve-e-et,"
    print "Al op de tafel gezet.\n"


print "Dit is het eerste couplet."
printTekst()
print "Dit is het tweede couplet."
printTekst()
print "Dit is het derde couplet."
printTekst()
```

Calling functions from functions

```
def print_lyrics():
    print "I'm a lumberjack, and I'm okay."
    print "I sleep all night and I work all day."


def repeat_lyrics():
    print_lyrics()
    print_lyrics()
    print_lyrics()
    print_lyrics()


repeat_lyrics()
```

Move this line to
the start of the script
and run it.
See what happens

In- and output

```
def waterNeeded (flour):
    '''
    Description: calculates amount of water for bread
    In: flour (grammes)
    Out: water (ml)
    '''
    water = flour * 325 / 500.0
    return(water)


flour = 1000
print "You need", waterNeeded(flour), "ml water"
```

```
# Standard recipe bread
# 500g flour
# 15g fresh yeast
# 10g sugar
# 10g salt
# 325ml water
```

Multiple arguments

```
def addTwo (a,b):
    added = a + b
    return added


x = addTwo(3,5)
print x
```

How much bread can I make?

- You have a certain amount of flour, yeast, sugar, salt and water
- Calculate how many breads you can make
- Hints:
  - Create a function
  - Do the calculations
  - Use if statements

```
# Standard recipe 1 bread
# 500g flour
# 15g fresh yeast
# 10g sugar
# 10g salt
# 325ml water
```

Why functions?

Easier to read and understand

Smaller, eliminate repetitive code

Easier to debug

You can reuse it, also in other programs

Next week

- Next: Loops and strings
- More programming exercise?
  - Chapter 3 and 4 of the book
- Ciao!!