Script language: Python

Control structures

Cédric Saule

Universität Bielefeld

12. Februar 2016

Python

–

Control structures

Control structures

- Command line parameters

Control structures

- Command line parameters
- Case distinction
  - if, elif, else
  - Conditional expressions

Control structures

- Command line parameters
- Case distinction
  - if, elif, else
  - Conditional expressions
- Loop
  - while
  - for

Command line parameters

- Importation of the module 'system' $\rightarrow$ access to system's components.
  `import sys`
- sys.argv contains command line parameters.
- sys.argv[0] contains the script call.
- sys.argv[1..] contains all parameters.

Exercise

Write a script 'echo.py' which behaves like the shell command 'echo'.

if, else

```
if <Condition>:
    Instructions
    ...
    Instructions
else:
    Instructions
    ...
    Instructions
```

elif

- Frequent constructs:

```
if <Condition1>:
    Instruction1
if <Condition2>:
    Instruction2
```

elif

- Frequent constructs:

```
if <Condition1>:
    Instruction1
if <Condition2>:
    Instruction2
```

- $\rightarrow$ **expensive**

elif

- Frequent constructs:

```
if <Condition1>:
    Instruction1
if <Condition2>:
    Instruction2
```

- $\rightarrow$ **expensive**
- better : elif

```
if <Condition1>:
    Instruction1
elif <Condition2>:
    Instruction2
```

Conditional expressions

- Alternative to if, else intruction
- Example:

```
if a == 1:
    b = 10
else:
    b = 1000
```

will become

```
b = (10 if a == 1 else 1000)
```

- Elegant (but unreadable) code
- Lazy evaluation

Exercise

Implement a script which waits for three arguments from the command line. The first argument is one of the following arithmetic operators [+,-,*,/] which will be applied to the next arguments. Result should be output in STDOUT.
Example:

```
> count.py - 1 2
> -1
> count.py / 20 4
> 5
```

while

- ```
  while <Condition>:
      Instruction
      ...
      Instruction
  ```
- Instructions are running **while** the condition is fulfilled.
  ```

break

- Interupts the loop.
- ```
  while <Condition>:
      ...
      if (Condition2):
          break
  ```
  ```

continue

- Interupts **a single** iteration
- ```
  while <Condition1>:
      ...
      if <Condition2>:
          continue
      ...
  ```
  ```

else

- Is executed **once**
- ```
  while <Condition>:
      ...
  else:
      Instruction
  ```
- Useful in connection with break
  ```

for

- `for <var> in <Object>:`
  `    Instruction`
  `    ...`
  `    Instruction`
- Object must be *iterable*
  $\rightarrow$ *Module/Object*
- `break`, `continue` or `else` usable

for - Example

- ```
  for i in range(5):
      print i
  ```

  `range` is a built-in function

- ```
  for c in ``Hello World'':
      print c
  ```

  An object string is *iterable*.

Read from STDIN

- Simple user input via STDIN
- Input([prompt]) -> value
- Typical instruction:
  `year = input(‘‘Birth year ? ’’)`

Exercise

Change the previous exercise so that when it is called without arguments, the script ask for the arithmetic operations and the two arguments from STDIN.