Lecture 2 – Program flow, Conditionals, Loops

https://cw.fel.cvut.cz/wiki/courses/be5b33prg/start

Tomas Jenicek
Czech Technical University in Prague,
Faculty of Electrical Engineering, Dept. of Cybernetics,
Center for Machine Perception

http://cmp.felk.cvut.cz/~jenicto2/
tomas.jenicek@fel.cvut.cz
• Problem formulation **(input / output)**

• Formalism **(math?)**

• Algorithm **(steps)**

• Implementation **(engineering)**

• Testing **(are we good?)**
• We use variables to **remember** things!
• The assignment statement gives a value to a variable
• Do not confuse = and == !
  
  = is **assignment** token such that *name_of_variable* = *value*
  
  == is operator to **test equality**
• Key property of a variable that **we can change its value**
• Naming convention: **with freedom comes responsibility!**
cannot begin with a number

>>> 76trombones = "big parade"
SyntaxError: invalid syntax

this $ is illegal character

>>> more$ = 1000000
SyntaxError: invalid syntax

class is reserved keyword

>>> class = "Computer Science 101"
SyntaxError: invalid syntax

• The longer life the longer name: very_long_name_of_my_var
• The more important the longer name
• Meaningful name does not add the meaning just by itself, the code must do this!
• Illegal name causes a syntax error
• Capitals: Variable vs variable
Python keywords have **special** purpose
- Always choose names **meaningful** to human readers
- Use **comments (#)** and **blank lines** to improve readability

---

source [http://openbookproject.net/thinkcs/python/english3e/variables_expressionsStatements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressionsStatements.html)
## BUILT-IN FUNCTIONS

<table>
<thead>
<tr>
<th>Function</th>
<th>Built-in Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs()</td>
<td>dict()</td>
</tr>
<tr>
<td>all()</td>
<td>hex()</td>
</tr>
<tr>
<td>any()</td>
<td>id()</td>
</tr>
<tr>
<td>ascii()</td>
<td>input()</td>
</tr>
<tr>
<td>bin()</td>
<td>int()</td>
</tr>
<tr>
<td>bool()</td>
<td>eval()</td>
</tr>
<tr>
<td>bytes()</td>
<td>exec()</td>
</tr>
<tr>
<td>bytearray()</td>
<td>filter()</td>
</tr>
<tr>
<td>callable()</td>
<td>format()</td>
</tr>
<tr>
<td>chr()</td>
<td>frozenset()</td>
</tr>
<tr>
<td>classmethod()</td>
<td>getattribute()</td>
</tr>
<tr>
<td>compile()</td>
<td>globals()</td>
</tr>
<tr>
<td>complex()</td>
<td>hasattribute()</td>
</tr>
<tr>
<td>delattr()</td>
<td>hash()</td>
</tr>
</tbody>
</table>

- Built-in functions have **special** purpose
- Study [https://docs.python.org/3.4/library/functions.html](https://docs.python.org/3.4/library/functions.html)

source: [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
• Integers (int) 1, 10, 124
• Strings (str) "Hello, World!"
• Float (float) 1.0, 9.999

• Strings in Python can be enclosed in either single quotes (') or double quotes ("), or three of each (''' or ''''')

source [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
• **OPERAND OPERATOR OPERAND**

• Operators are **special tokens** that represent computations like addition, subtraction, multiplication, division etc.

• The values the operator uses are called **operands**

• When a variable name appears in the place of an operand, it is replaced with its value before the operation is performed

• Division `/` vs *floor* division `//`

source [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
• Evaluation depends on the rules of precedence:
  1. Parentheses (for order, readability)
  2. Exponentiation
  3. Multiplication and Division
  4. Addition and Subtraction
• Order **left-to-right** evaluation on the same level, with the exception of exponentiation (***)

source [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
• The modulus operator works on **integers** (integer expressions)
• Definition: modulus is the **remainder** when the first number is divided by the second
• Modulus operator is a percent sign `%`
• Syntax is the same as for other operators
• The same **precedence** as the **multiplication** operator

source [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
Functions, `int()`, `float()` and `str()` convert their arguments into types `int`, `float` and `str` respectively.

- The type converter `float()` can turn an integer, a float, or a syntactically legal string into a float.
- The type converter `str()` turns its argument into a string.
- One symbol can have different meaning depending on the data type(s) - try & explore & understand.

Source: [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
• You cannot perform mathematical operations on strings, even if the strings look like numbers
• The `+` operator represents **concatenation**, not addition
• The `*` operator also works on strings; it performs **repetition** (one of the operands has to be a string; the other has to be an integer)
• Built-in function to get input from a user:

```python
input("Message to the user!")
```

• User input is stored as string

• Combine with type conversion

source [http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html](http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html)
• Combination of the elements of a program: variables, expressions, statements, and function calls
• One of the most useful features of programming languages
• Take small building blocks and compose them into larger chunks

source: http://openbookproject.net/thinkcs/python/english3e/variables_expressions_statements.html
• The variable `friend` at line 1 is the **loop variable**
• Lines 2 and 3 are the **loop body**
• The loop body is **always indented**
• The indentation determines exactly what statements are “in the body of the loop”
• At the end of each execution of the body of the loop, Python returns to the `for` statement, to see if there are more items to be handled, and to assign the next one to the loop variable

source: [http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html](http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html)
On each iteration or pass of the loop:

- Check to see if there are still more items to be processed
- If there are none left (the terminating condition of the loop) the loop has finished
- If there are items still to be processed, the loop variable is updated to refer to the next item in the list
- Program execution continues at the next statement after the loop body
- To explore: early break, or for – else loop

source: http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html
• Control flow (control of the flow of execution of the program)
• As program executes, the interpreter *always keeps track* of which statement is about to be executed
• Control flow until now has been strictly *top to bottom*, one statement at a time, the *for loop* changes this!

source [http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html](http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html)
CONDITIONAL EXECUTION

```python
if BOOLEAN_EXPRESSION:
    STATEMENTS_1   # Executed if condition evaluates to True
else:
    STATEMENTS_2   # Executed if condition evaluates to False
```

- Condition **IF – ELSE**
- Conditional statement – the ability to check conditions and change the behavior of the program accordingly

source [http://openbookproject.net/thinkcs/python/english3e/conditionals.html](http://openbookproject.net/thinkcs/python/english3e/conditionals.html)
• Condition **IF only**
• No ELSE statement
• To control flow only for specific condition

```python
if x < 0:
    print("The negative number ", x, " is not valid here.")
x = 42
print("I've decided to use the number 42 instead.")
print("The square root of ", x, "is", math.sqrt(x))
```
• Condition chaining
  **IF – ELIF – ELSE**

• Recommendation: handle all distinctive options by separate condition, use else to handle all other

source [http://openbookproject.net/thinkcs/python/english3e/conditionals.html](http://openbookproject.net/thinkcs/python/english3e/conditionals.html)
Nesting conditions builds hierarchy of decisions (decision trees)

Nesting may reduce readability and clarity

source: [http://openbookproject.net/thinkcs/python/english3e/conditionals.html](http://openbookproject.net/thinkcs/python/english3e/conditionals.html)
• Early return / early break
• Can be used to speed-up code execution
• Special condition: FOR – ELSE

• Test conditions and change the program behavior depending on the outcome of the tests
• Boolean value is either True or False
• Named after the British mathematician, George Boole, who first formulated Boolean algebra

source http://openbookproject.net/thinkcs/python/english3e/conditionals.html
Boolean expression is an expression that evaluates to produce a result which is a **Boolean value**

Six common **comparison operators** which all produce a bool result (different from the mathematical symbols)
three logical operators, **and**, **or**, and **not**, that allow to build more complex expressions from simple Boolean expressions

semantics (**meaning**) of these operators is similar to natural language equivalent
Short-circuit evaluation:

- **OR** – if the expression on the left of the operator yields *True*, Python does not evaluate the expression on the right.
- **AND** – if the expression on the left yields *False*, Python does not evaluate the expression on the right.
- **Truth table** – list of all the possible inputs to give the results for the logical operators.

source [http://openbookproject.net/thinkcs/python/english3e/conditionals.html](http://openbookproject.net/thinkcs/python/english3e/conditionals.html)
Each of the six relational operators has a **logical opposite**

- Recommendation: **not** operators may reduce readability, use logical opposites instead
\[ n \times 0 = 0 \]

\[
\begin{align*}
&\text{x and False} = \text{False} \\
&\text{False and x} = \text{False} \\
&\text{y and x} = \text{x and y} \\
&\text{x and True} = \text{x} \\
&\text{True and x} = \text{x} \\
&\text{x and x} = \text{x}
\end{align*}
\]

\[
\begin{align*}
&\text{x or False} = \text{x} \\
&\text{False or x} = \text{x} \\
&\text{y or x} = \text{x or y} \\
&\text{x or True} = \text{True} \\
&\text{True or x} = \text{True} \\
&\text{x or x} = \text{x}
\end{align*}
\]

\[
\text{not (not x)} = x
\]

source [http://openbookproject.net/thinkcs/python/english3e/conditionals.html](http://openbookproject.net/thinkcs/python/english3e/conditionals.html)
De Morgan’s laws rules allow the expression of **conjunctions** and **disjunctions** in terms of each other via **negation**

**Example**: suppose we can slay the dragon only if our magic sword is charged to 90% or higher **and** we have 100 or more energy units in our protective shield.
DE MORGAN’S LAWS

• Example: suppose we can slay the dragon only if our magic sword is charged to 90% or higher and we have 100 or more energy units in our protective shield

```python
if (sword_charge < 0.90) or (shield_energy < 100):
    print("Your attack has no effect, the dragon fries you to a crisp!")
else:
    print("The dragon crumples in a heap. You rescue the gorgeous princess!")
```

```python
if (sword_charge >= 0.90) and (shield_energy >= 100):
    print("The dragon crumples in a heap. You rescue the gorgeous princess!")
else:
    print("Your attack has no effect, the dragon fries you to a crisp!")
```
• **Example**: complete the table ..

```plaintext
<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>r</th>
<th>(not (p and q)) or r</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>?</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>?</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>?</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>?</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>?</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>?</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>?</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>?</td>
</tr>
</tbody>
</table>
```

source [http://openbookproject.net/thinkcs/python/english3e/conditionals.html](http://openbookproject.net/thinkcs/python/english3e/conditionals.html)
This lecture re-uses selected parts of the OPEN BOOK PROJECT
Learning with Python 3 (RLE)
http://openbookproject.net/thinkcs/python/english3e/index.html
available under GNU Free Documentation License Version 1.3

• Version date: October 2012
• by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers (based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)
• Source repository is at https://code.launchpad.net/~thinkcspy-rle-team/thinkcspy/thinkcspy3-rle
• For offline use, download a zip file of the html or a pdf version from http://www.ict.ru.ac.za/Resources/cspw/thinkcspy3/