Selection statements

CSE 1310 – Introduction to Computers and Programming
Alexandra Stefan
University of Texas at Arlington
Book reference

  - Second hand book is ok (the book code is not needed)

- Even though the slides may show examples from the book at times, they do not attempt to cover the book.
  - Book reading is not required, but **HIGHLY recommended**.

- Book chapter for Selection statements:
  - Chapter 2:
    - 2.1 (2.1.1-2.1.3) : if, if-else, and examples,
    - 2.2.1, 2.2.2, 2.2.4 - 2.2.6): boolean expressions
    - 2.2.8, 2.2.9 : if-elif-else
  - Other topics:
    - section 2.2.7 (multiple assignments, variable swap)
    - section 'What does it mean to be equal‘ of 2.2.3 (at page 105)
The *if* statement

The *if* statement allows the program to decide whether or not a certain group of instructions will be executed.

**Syntax:**

```python
if condition:
    statement
    statement
    ...
    statement
```

*Block of instructions or suite*

(The indentation defines the group)
The *if* statement

The if statement allows the program to decide whether or not a certain group of instructions will be executed.

Syntax:

```python
if condition:
    statement
    statement
    ...
    statement
```

*Block of instructions or suite*

(The indentation defines the group)

Example:

```python
x = int(input('x = '))
if x < 0:
    print(x, 'is negative')
```
The *if* statement

Example:

```python
x = int(input('x = '))
if x < 0 :
    print(x , 'is negative')
```

- Indentation groups the statements together. Typically 4 white spaces are used for indenting.
- Be consistent when specifying the indentation: do not mix spaces and tabs
- How many instructions are in this program?
- Where will the next instruction be written (e.g. What if you want to print ‘Bye’ at the end of your program)?
The *if*-else statement

The if-else statement allows the program to choose between two groups of instructions. It will execute one, or the other (never both, never none).

```python
if condition :
    statement
    statement
    ....
    statement
else :
    statement
    statement
    ....
    statement
```

- Where will the next instruction be written?
- Draw the flow of control diagram.
The *if-else* statement

Example:

```python
x = int(input('x = '))
if x < 0:
    print(x, 'is negative')
else:
    print(x, 'is positive')
```
Practice

• Can you think of a program that would require the use of:
  – *if*
  – *if-else*

• Leap year: divisible by 4, except 100, ok 400
• Min of 2 or 3 numbers
• Implement the absolute value functionality
• Write a program that computes the letter grade based on the score (according to the standard scheme: 90,80,70,60)
Let’s take the *if* statement apart!

- Identify and understand every piece of syntax

```python
if condition:
    statement
    ...
    statement

x = int(input('x = '))
if x < 0:
    print(x, 'is even')
    print('second print')

print('third print')
```
More on the *condition* of an *if* statement

- What is the *condition* of an *if* statement?
  - Is it an expression or a statement?
  - Can you give examples of other conditions?
  - What kind of data can you use in place of the condition? A number? A string?
    - Only boolean expressions
    - Be aware of Python’s automatic conversions!
The Bool Type

• Expressions of type boolean (bool) can only have two values: True, or False.
  – True and False are reserved keywords in Python.

```python
>>> 3 > 2
True
>>> type(True)
<type 'bool'>
```

Preferred style (parenthesize)

```python
>>> a = 3 == 2
>>> a
False
>>> a = (3 == 2)
>>> a
False
```

```python
>>> b = 4
>>> a = (b != 5)
>>> a
True
```
Boolean expressions

• Building boolean expressions:
  – Using relational operators:
    • $3 < 4$
    • ‘<‘ is called a relational operator.
    • Can you think of other relational operators?
  – Other operators to build boolean expressions?
    • What if you want to check for two conditions at the same time? (E.g.: It’s week-day and between 8-10am?)
      How will you connect them?
    • Boolean operators (also called logical operators)
Relational operators

• The following operators generate boolean results:

  ==  equality
  !=  not equal
  <   less than
  >   greater than
  <=  less than or equal to
  >=  greater than or equal to
Boolean Operators

• The following Boolean operators can be used to produce Boolean expressions:

  - *not* (highest)
    
    \[
    \text{not } A \quad \text{and } B \quad \text{or } C
    \]
    
    - \((\text{not } A) \quad \text{and } B \quad \text{or } C\)
    
  - *and* (middle)
    
    \[
    \text{not } A \quad \text{or } B \quad \text{and } C
    \]
    
    - \((\text{not } A) \quad \text{or } (B \quad \text{and } C)\)
    
  - *or* (lowest)
    
    \[
    A \quad \text{or } B \quad \text{not } C
    \]
    
    - \((A \quad \text{or } (\text{not } B) \quad \text{and } C)\)

• Make the truth tables for these operators:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A or B</th>
<th>A and B</th>
<th>not A</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
## Boolean Operators

### Simple boolean expressions

```python
>>> a = 3
>>> b = 4

>>> (a == b) or (a+b == 7)
True

>>> (a == b) and (a+b == 7)
False

>>> not(a == b)
True
```

```python
>>> n = 10
>>> n == 10,20
(True, 20)

>>> n == 10 or 20
True

>>> n == 20 or 10
10

>>> 20 or 10
20

>>> False or 10
10

>>> True or 10
True
```

### Be aware of commas!

```python
>>> n = 10
>>> n == 10,20
(True, 20)

>>> n == 10 or 20
True

>>> n == 20 or 10
10

>>> 20 or 10
20

>>> False or 10
10

>>> True or 10
True
```
Combining operators

• What does this line do?

```python
>>> (3 == 5) and (2 < 3) or (3 >= 0)
```

- Evaluate the parenthesis first.
- After that, what will you evaluate (the ‘and’ or the ‘or’)?
Combining operators

• What does this line do?

```python
>>> (3 == 5) and (2 < 3) or (3 >= 0)
```

• Use parentheses to make the meaning of these statements clear.

```python
>>> ((3 == 5) and (2 < 3)) or (3 >= 0)  # True
>>> (3 == 5) and ((2 < 3) or (3 >= 0))  # False
```
Chained relational operators

```python
>>> 0 <= x <= 5
Is : (0<=x) and (x <= 5)
```

- Be careful in using them!:
  ```python
  >>> 4 < 5 == True  # You may think it should be True
  Is: (4<5) and (5 ==True)  # It evaluates to False
  ```
One more selection statement: 

*if-elif-else*
The *if-elif-else* statement

```python
if condition1:
    # suite
elif condition2:
    # suite
elif condition3:
    # suite
...
else:
    # suite
```

• If `condition1` evaluates to True the other ones will not be evaluated. Compare this with multiple `if` statements (at the same level).
• It is ok both with and without the last `else`. Is there any difference between the two?
• Draw the flow of control diagram.
Practice

• Construct boolean expressions (compute the tables)

• Write examples that use conditional statements.
What does it mean to be equal?

• 2\textsuperscript{nd} edition book, page 105
Float numbers are approximations of real numbers

```python
>>> u = 11111113
>>> v = -11111111
>>> w = 7.51111111

>>> (u + v) + w
9.51111111

>>> u + (v + w)
9.5111111044837
```
Float numbers are approximations of real numbers

>>> u = 11111113
>>> v = -11111111
>>> w = 7.51111111

>>> x = (u + v) + w
>>> y = u + (v + w)

>>> abs(x-y)<0.0000001  # abs computes the absolute value
Value versus ID
What does it mean to be equal?

Value versus ID

• Two different variables point to objects that have the same value.

• Two different variables point to the same object (same ID).
  – Use the `is` function to see if two variables point to the same object.

• Use the `id` function to get the object ID.
What does it mean to be equal?

Book example:

```python
>>> a_float = 2.5
>>> b_float = 2.5
>>> c_float = b_float
```
What does it mean to be equal?

Book example:

>>> a_float = 2.5
>>> b_float = 2.5
>>> c_float = b_float

>>> id(a_float)
>>> id(b_float)
>>> id(c_float)

>>> a_float == b_float
>>> b_float == c_float

>>> a_float is b_float
>>> b_float is c_float