**Learning outcomes**

- At the end of this lecture, students should:
  - be able to evaluate a boolean expression
  - be familiar with boolean values True and False
  - be able to use conditional statements (if, else)
  - be able to use relational operators (> , < , <= , >= and ==)

---

**Boolean expressions**

- A condition is an expression which evaluates to either True or False
- An expression which evaluates to either True or False is called a **boolean expression**.
- In Boolean expressions, relational operators are used to compare values
- The relational operators are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>

**Boolean variables**

- Variables can be used to store the result of a comparison, i.e., to store a boolean expression
- For example:

```python
def main():
    exam_mark = 76
    age = 8
    points_so_far = 56

    passed_exam = exam_mark >= 50
    has_won_game = points_so_far > 70
    is_old_enough = age > 10
    is_old_enough = passed_exam != has_won_game

    print(passed_exam, has_won_game, is_old_enough)
main()
```

True False True
Controlling the flow of execution

- In all the programs written so far, the statements inside functions are executed in the order in which they are written, e.g., all the statements in the main() function are executed and they are executed sequentially.

- We would like to be able to control the execution of our code so that blocks of code are only executed under certain conditions.

- Control structures allow us to change the flow of statement execution in our programs.

Python syntax for an if statement

- In an if statement (selection statement) the code in the if block is executed if the condition is true.

```
import random
def main():
    num_odds = 0
    num1 = random.randrange(0, 100)
    if num1 % 2 == 1:
        num_odds += 1
    num2 = random.randrange(0, 100)
    if num2 % 2 == 1:
        num_odds += 1
    num3 = random.randrange(0, 100)
    if num3 % 2 == 1:
        num_odds += 1
    print(num1, num2, num3)
    print("ODD NUMBERS:", num_odds)
main()
```

```python
40 71 41
ODD NUMBERS: 2
```
Boolean expressions – logical operators

- As well as the relational operators, we can use the following logical operators in Boolean expressions:

  - `and`
  - `or`
  - `not`

- The three truth tables for these logical operators are shown below:

  ![Truth Tables](image)

Logical operators - examples

- Assume that the variable, `value`, has been initialised.

  **Is value greater than 10 and less than 100**
  \[\text{value} > 10 \text{ and } \text{value} < 100\]

  **Is value greater than or equal to 10 or is the value equal to 5**
  \[\text{value} >= 10 \text{ or } \text{value} == 5\]

  **Is value not greater than 8**
  \[\text{not value} > 8\]

  **Is value not greater than 8 and not equal to 1**
  \[\text{value} <= 8 \text{ and } \text{value} != 1\]

Use parentheses in boolean expressions

- Use parentheses to group sections of your boolean expressions to make the program more readable, e.g.,

  \[\text{a} > \text{b} \text{ or } (\text{a} > \text{c} \text{ and } \text{b} < 45)\]

  is more readable than

  \[\text{a} > \text{b} \text{ or } \text{a} > \text{c} \text{ and } \text{b} < 45\]

  **not as clear**

  but do not overload your boolean expressions with unnecessary parentheses, e.g.,

  overuse of unnecessary parentheses

  \[(((\text{a} > \text{b}) \text{ or } ((\text{a} > \text{c}) \text{ and } (\text{b} < 45)))\]
If statements – a common mistake

Remember that the equality operator is ==.
What is the problem with the code below?

```python
def main():
    val1 = 50
    val2 = 53
    if val1 == val2 - 3:
        print("Unbelievable")
main()
```

Note: single = is the assignment operator.

Comparing float numbers

Floating point numbers are always stored approximately. It is dangerous to test doubles for equality using ==.

```python
val1 = 0.3
val2 = 0.1 * 3
if val1 == val2:
    print("Sigh!")
if val1 != val2:
    print("maybe yes, maybe no!")
```

Test equality of floats by accepting all values within an acceptable error limit:

```python
val1 = 0.3
val2 = 0.1 * 3
error_limit = 0.00001
if abs(val1 - val2) < error_limit:
    print("Close enough!")
```

Examples of Python features used in this lecture

```python
exam = exam_mark / 100 * 60
test = test_mark / 100 * 60
passed_theory = exam + test >= 50

number = 32
if number % 6 < 2:
    number += 1

val1 = 0.3
val2 = 0.1 * 3
error_limit = 0.00001
if abs(val1 - val2) < error_limit:
    print("Close enough!")
```