1DV501: Introduction to programming

Föreläsning 3

Boolean Expressions and If-statements

Agenda

- Boolean Values
- Boolean Expressions
- Logical Operators
- If-statements
- String indexing (extra material)
- Random number generators (extra material)

- Reading Instructions: Sections 4.1-4.14 in textbook by Halterman
- Exercises: Exercises 13-18 in Assignment 1

Control Statements

By using control statements the program can choose one execution path out of several possible options or it can repeat a sequence of statements several times.

Until Now:

- Sequential execution (from the top and downwards)

Control Statements:

- Selective statements: *Choose one execution path out of several possible options*
- In Python: if- statements
- Iterative (or loop) statements: *Repeat a sequence of statements several times*
- In Python: while- and for-statements (Next lecture!)

A first example

Assign a Swedish grade (Fail, Pass, or Pass with distinction) to an exam result python

```
MIN, MAX, PASS, VG  = 0, 100, 50, 75
points = int( input("Enter exam result: ") )

if points >= 0 and points < PASS:
    print("Fail")
elif points >= PASS and points < VG:
    print("Pass")
elif points >= VG and points <= MAX:
    print("Pass with distinction - Very Good!")
else:
    print("Invalid exam result: ", points)
```

Output:

```
Enter exam result: 78
Pass with distinction - Very Good
```

Details from this example will be discussed in the following slides

In [2]:

```
MIN, MAX, PASS, VG  = 0, 100, 50, 75
points = int( input("Enter exam result: ") )

if points >= 0 and points < PASS:
    print("Fail")
elif points >= PASS and points < VG:
    print("Pass")
elif points >= VG and points <= MAX:
    print("Pass with distinction - Very Good!")
else:
    print("Invalid exam result: ", points)
```

```
Pass
```

Boolean Variables

```
a = True
print(a, type(a))
```

```
True <class 'bool'>
```

```
b = False
print(b, type(b))
```

```
False <class 'bool'>
```

In [4]:

```
a = True
print(a, type(a))
```

```
True <class 'bool'>
```

Boolean Expressions

```
a = 10 < 7
print(a, type(a))
```

```
False <class 'bool'>
```

```
b = 8 != 4
print(b, type(b))
```

```
True <class 'bool'>
```

In [5]:

```
a = 10 < 7
print(a, type(a))
```

```
False <class 'bool'>
```

In [6]:

```
8 is not 4
```

Out[6]:

```
True
```

In [ ]:

```

```

- A boolean variable (typ e bool) can only take the values True or False
- We can generate boolean values using comparison op erators like < or != ...
- ... or by using logical operators like `and` or `not`

In [7]:

```
b = 1
c = 2

b < 10 and c < 10
```

Out[7]:

```
True
```

In [8]:

```
b != 2
```

Out[8]:

```
True
```

Boolean expressions

```
if points >= 0 and points < 50:
    print("Fail")
```

- `points >= 0 and points < 5` is a so-called boolean expression
- Boolean expression returns the values `True` or `False`
- In an `if` -statement, the statements after the boolean expression is only executed if the value of the boolean expression is `True` .

A boolean expression usually consists of:

1. Numerical values (such as 17, -100, 3.14) or numeric variables (such as MAX, PI, grade)
2. Comparison-operators: `<, <=, >, >=, ==, !=`
3. Logical operators: `and, or, not`
4. Functions returning boolean (Not in this lecture)

- **NOTE** The result of a comparison (such as `points >= 0`) is `True` or `False`) consider each boolean expression to be an assertion.

In [12]:

```
1 = 1.1
```

```
  File "<ipython-input-12-074bbff55c4e>", line 1
    1 = 1.1
          ^
SyntaxError: can't assign to literal
```

Comparison Operators

- Python **comparison operators** (also called **relational operators**)

Expression	Meaning
`x == y`	`True` if x = y, else `False`
`x > y`	`True` if x > y, else `False`
`x < y`	`True` if x < y, else `False`
`x <= y`	`True` if x = y or x < y, else `False`
`x != y`	`True` if x not equal to y, else `False`

- **NOTE** Notice that `!=` means "*not equal to*" and `==` means "*equal to*". Do not mix up the comparison operator `==` with the standard assignment operator `=`

Logical Operators

- **`and`** :

  -> `A` and `B` is `True` if both `A` and `B` are `True` , otherwise it is `False`
- **`or`** :

  -> `A` and `B` is `True` if at least one of `A` and `B` is `True` , otherwise it is `False`
- ***`not`*** :

  -> `not A` negates the logical value, that is `not A` is `True` if `A` is `False` , and the other way around.

```
A       B       A and B       A or B      not A
====    ====    ========     ========     ====
true    true      true         true       false
true    false     false        true
false   true      false        true       true
false   false     false        false
```

NOTE Logical operators can only be applied on Boolean values. Expressions like `x>1 or 7` gives an error since 7 is not a boolean value.

Tasks: True or false?

In [64]:

```
a = False
b = False
a and b
```

Out[64]:

```
False
```

In [17]:

```
( 12 > 10 ) and ( 9 < 6 )
```

Out[17]:

```
False
```

In [14]:

```
5 > 4 or 8 < 6
```

Out[14]:

```
True
```

In [15]:

```
7 > 4 or 12 > 8 and 4 < 8
```

Out[15]:

```
True
```

In [18]:

```
6 > 3 and not (5 < 3) and not not (8 > 3)
```

Out[18]:

```
True
```

In [20]:

```
(10 + 20) < (3 + 4 * 5)
```

Out[20]:

```
False
```

In [21]:

```
10 == 20 or 3 + 4 > 5
```

Out[21]:

```
True
```

In [22]:

```
10 != 20 and not (7>5) or 5 >=5
```

Out[22]:

```
True
```

In [23]:

```
# Not recommended

10 != 30 and not not not ...
not 4 < 2 or not 4 > 2 and ...
not not not not...
not not 1 != 0 or...
5 < 10
```

Out[23]:

```
True
```

In [ ]:

```

```

- 10 + 20 < 3 + 4 *5* (False since 30 > 23)*
- 10 == 20 or 3 + 4 > 5 *(True since 7>5 is true)*
- 10 != 20 and not (7>5) or 5 >=5 *(True and False or True ==> True)*

- Different operators have different priorities.
- The operators with highest priorities are computed first.
- By using parentheses you can change the order. Ex: `3+4*5` is not equal to `(3+4)*5`
- Numerical operators: `*,/` are computed before `+,-`
- Logical operators: `not` before `and` before `or`

    1. `not`
    2. `and`
    3. `or`

- Generally: `NumOP > CompOP > LogOP`

Grades Example Revisited

We should now be able to understand all boolean expressions in the grade example

In [25]:

```
MIN, MAX, PASS, VG  = 0, 100, 50, 75
points = int(input("Enter exam result: "))

if points >= 0 and points < PASS:
    print("Fail")
elif points >= PASS and points < VG:
    print("Pass")
elif points >= VG and points <= MAX:
    print("Pass with distinction - Very Good!")
else:
    print("Invalid exam result: ", points)
```

```
Invalid exam result:  200
```

If-statements

Example: Simple if statement:

```
print("Computes A divided by B (A/B)")
a = int(input("Please enter A: "))
b = int(input("Please enter B: "))

if b != 0:
    print(a, "/", b, "=", a/b)
```

In [27]:

```
print("Computes A divided by B (A/B)")
a = int(input("Please enter A: "))
b = int(input("Please enter B: "))

if b != 0:
    print(a, "/", b, "=", a/b)
```

```
Computes A divided by B (A/B)

5 / 3 = 1.6666666666666667
```

In [28]:

```
5/0
```

```
---------------------------------------------------------------------
----
ZeroDivisionError                         Traceback (most recent call l
ast)
<ipython-input-28-0106664d39e8> in <module>
----> 1 5/0

ZeroDivisionError: division by zero
```

- Executes the statement `print(a, "/", b, "=", a/b` only if the boolean condition `b !=` evaluates to `True`
- b = 0 --> no output

Example: Simple if-else statement

In [30]:

```
print("Computes A divided by B (A/B)")
a = int( input("Please enter A: ") )
b = int( input("Please enter B: ") )

if b == 0:
    print("B must not be zero!")
else:
    print(a, "/", b, "=", a/b)
```

```
Computes A divided by B (A/B)

B must not be zero!
```

- if branch `print("B must not be zero!")` only if `b = 0` , ...`
- ...otherwise (in all other cases) uses the else branch `print(a, "/", b, "=", a/b)`
- One branch (if or else) is always executed

---

if-else in general

```
if "condition":
    "if body"
else:
    "else body"
```

- The reserved word `if` begins the if-else statement`
- A colon `(:)` must follow the condition`
- The reserved word `else` begins the second part of the `if/else` statement.
- A colon `(:)` must follow the else.

Rules for indentation

```
# OK!
if n == 1:
    print("One")

# OK!
if n == 1: print("One")

# Not OK!
if n == 1:
print("One")
```

```
# OK!
if n == 1:
    n = n + 1  # indent 4
    print("Increasing n")

# OK!
if n == 1:
  n = n + 1  # indent 2
  print("Increasing n")

# Not OK!
if n == 1:
  n = n + 1  # indent 2
    print("Increasing n") # indent 4
```

In [36]:

```
n = 1

if n == 1:
    print('ett')
    print('två')
```

```
ett
två
```

In [ ]:

```
n=1
if n == 1:
                          print(n)
```

In [ ]:

```
n=1
if n == 1:
                                                    print(n)
```

- The content of an if (or else) body is defined by indentations
- All statements must have the same indentation
- The bodies must be indented *at least one step*. Some programmers consistently use two, but four is the most popular step size (VSCode tab key gives by default four steps use it!)

Multi-choice using if-elif-else

- if-elif-else allows us to chose one out several option`
- The keyword `elif` is a short version of "else if"
- Always starts with `if` "condition": ...
- ... followed by any number of `elif` "condition"
- ... followed by a single `else:`
- The use of `else:` is optional in an if-elif statement
- Only the first branch that evaluates to `True` is executed

In [40]:

```
n = int( input("Please enter a positive integer: ") )

if n < 1:
    print("The number must be positive!")
elif n == 1:
    print("One")
elif n == 2:
    print("Two")
elif n == 3:
    print("Three")
else:
    print("A number larger than three: ", n)
```

```
A number larger than three:  4
```

Python Simplifications

- Python allows us to simplify certain boolean expressions

In [43]:

```
points = 20
# From the grades example
if points >= 0 and points < PASS:
    print("Fail")
```

```
Fail
```

In [44]:

```
# Equivalent and simpler
if 0 <= points < PASS:
    print("Fail")
```

```
Fail
```

In [45]:

```
x, y, z = 1, 1, 1

# Are x,y,z all the same?
if (x == y) and (y == z):
    print("They are the same")
```

```
They are the same
```

In [46]:

```
# Equivalent and simpler
if x == y == z:
    print("They are the same")
```

```
They are the same
```

- **Case 1**: A logical expression `points >= 0 and points < PASS` is replaced with an interval `0 <= points < PASS`
- **Case 2**: Once again, a simplified expression that can be generalized to multiple variables `a == b == c == d == ....`

In [47]:

```
MIN, MAX, PASS, VG  = 0, 100, 50, 75
points = int( input("Enter exam result: ") )

if 0 <= points < PASS:
    print("Fail")
elif PASS <= points < VG:
    print("Pass")
elif VG <= points <= MAX:
    print("Pass with distinction - Very Good!")
else:
    print("Invalid exam result: ", grade)
```

```
Pass
```

Nestled example: Odd or even?

In [48]:

```
# Check if a number is even or odd
n = int(input("Please enter a positive integer:"))

if n < 1:  # Check if positive
    print("The number must be positive!")
else:
    if n%2 == 0: # Check if even
        print(n, "is an even number")
    else:
        print(n, "is an odd number")
```

```
7 is an odd number
```

- We have an if-else statement inside the else branch of an outer statement
- Statements inside other statements are called *nestled statement*

Nestled Statements

- Understanding each control statement by itself is rather easy
- Solving problem requiring only one such statement is also often rather easy
- However, many problems require multiple nestled control statements

```
if n > 0:
    if n % 2 == 0:
        ...
    else:
        while n > 10:
          ...
else:
    for i in range(2,6):
```

- Solution with nestled statements --> more complex --> more training needed
- Assignment 2 has a large set of problems that *require nestled statements*
- `while` and `for` statements will be presented in the next lecture

Conditional Expressions - A Python Shortcut

In [49]:

```
a, b = 3, -5
```

In [53]:

```
# Find smallest number
if a < b:
    min1 = a
else:
    min1 = b
print("Min is", min1)  # Prints -5
```

```
Min is -5
```

In [54]:

```
# Equivalent
min = a if a < b else b
print("Min is", min)
```

```
Min is -5
```

In [55]:

```
# Equivalent
print("Min is", a if a < b else b)
```

```
Min is -5
```

In [56]:

```
# Even or odd
print(a, "is", "even" if a % 2 == 0 else "odd")
```

```
3 is odd
```

Conditional Expressions (cont.)

Conditional expressions like

```
min = a if a < b else b

or

s = "even" if a % 2 == 0 else "odd"
```

- is a **short version** of an if-else statement.
- The general form is `"true_expression" if "condition" else "false_expression`

- It evaluates to `true_expression` if `condition` is `True`
- It evaluates to `false_expression` if `condition` is `False`
- **Warning:** Use it with care! It is likely to produce code that is hard to read and understand

String Indexing

```
s = "Hello Python"

# Characters at positions 0 and 6
print(s[0], s[6], type(s[0]))     # Output: H P <class 'str'>

sub = s[1:4]  # Positions 1 to 3
print(sub, type(sub))        # Output ell <class 'str'>

length = len(s)  # String length
print(length, type(length))     # Output: 12 <class 'int'>
```

In [57]:

```
s = "Hello Python"

# Characters at positions 0 and 6
print(s[0], s[6], type(s[0]))     # Output: H P <class 'str'>

sub = s[1:4]  # Positions 1 to 3
print(sub, type(sub))        # Output ell <class 'str'>

length = len(s)  # String length
print(length, type(length))     # Output: 12 <class 'int'>```
```

```
H P <class 'str'>
ell <class 'str'>
12 <class 'int'>
```

String Indexing

- `s[6]` select character at position 6
- **Warning:** Positions start at position zero \ra `s[0]` is the first character
- `s[1:4]` strings with characters 1 to 3
- **Warning:** First position (1) included, final position (4) **not** included
- The function `len(...)` gives the length of a string

Random Numbers

(NB. Pseudo-random, Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.)

```
import random  # Always at start of programs

n1 = random.randint(90,100)  # Random integer in interval [90,100]
n2 = random.randint(-10,10)
n3 = random.randint(-30,-20)
print(n1, n2, n3)

f1 = random.uniform(40, 50) # Random float in interval [40.0,50.0]
f2 = random.uniform(0, 1)
f3 = round( random.uniform(0, 10), 2) # Rounded to two decimals
print(f1, f2, f3)
```

- Random functions are not available by default they must be imported
- `import random` makes the `random` module available
- `random.randint(90,100)` call function `randint` in module `random`
- More about modules and imports later on ...

In [59]:

```
import random  # Always at start of programs

n1 = random.randint(90,100)  # Random integer in interval [90,100]
n2 = random.randint(-10,10)
n3 = random.randint(-30,-20)
print(n1, n2, n3)

f1 = random.uniform(40, 50) # Random float in interval [40.0,50.0]
f2 = random.uniform(0, 1)
f3 = round( random.uniform(0, 10), 2) # Rounded to two decimals
print(f1, f2, f3)
```

```
92 7 -30
46.92390214297626 0.9745740547622217 9.67
```

Programming Examples - Duplicates

Write a program *duplicates.py* which reads three integers from the keyboard and decides if they contain any duplicate elements or if they are all unique.

Execution examples:

```
Enter three integers A, B, C
Enter A: 2
Enter B: 5
Enter C: 5
We have duplicates!

Enter three integers A, B, C
Enter A: 4
Enter B: 6
Enter C: 8
They are all unique!
```

In [60]:

```
print("Enter three integers A, B, C")
a = int( input("Enter A: ") )
b = int( input("Enter B: ") )
c = int( input("Enter C: ") )

if a == b or b == c or c == a:
    print("We have duplicates!")
else:
    print("They are all unique!")
```

```
Enter three integers A, B, C

We have duplicates!
```

Programming Examples - Dividable

Write a program *dividable.py* which reads a positive integer from the keyboard and decides if it is dividable by 3 or 4, **but not both**.

Execution examples:

```
Please provide a positive integer: 8
8  is dividable by 4 (but not 3)

Please provide a positive integer: 12
12 does not fulfill the requirements

Please provide a positive integer: -7
The number must be positive!
```

In [62]:

```
### # Read user input
n = int( input("Please provide a positive integer:") )

if n < 0: # Check for positive
    print("The number must be positive!")
else:
    if n % 3 == 0 and n % 4 != 0: # by 3 but not by 4
        print(n, " is dividable by 3 (but not 4)")
    elif n % 4 == 0 and n % 3 != 0: # by 4 but not by 3
        print(n, " is dividable by 4 (but not 3)")
    else: # Not fulfilling requirements
        print(n, "does not fulfill the requirements")
```

```
8  is dividable by 4 (but not 3)
```

Course information

- Assignment 1 deadline: **Sunday September 13**

- Campus students must present their A1 solutions at the final tutoring session before the deadline
- Distance students will be informed about a video meeting around (short before or after) the deadline
- **Important:** Distance students must get in contact with their tutoring supervisor to register as an active distance students.

LNU Rule: Non-active students should be unregistered

We will consider a student as active if they:

- Campus students that show up and are active (present Assignment 1 solutions) at tutoring sessions, or
- Distance students that sign up for for video meeting to present their Assignment 1 solutions

In [ ]: