

Lecture 5

Boolean Expressions and Conditional Statements

Lecture 5 - Boolean Algebra

- What are we missing?
 1. READ account_balance
 2. READ purchase_amount
 3. IF account_balance >= purchase_amount THEN
 4. APPROVE payment
 5. ELSE
 6. REJECT payment
 7. END IF

Lecture 5 - Boolean Algebra

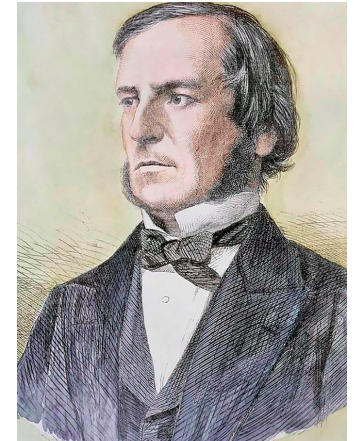
- What are we missing?
 1. READ account_balance
 2. READ purchase_amount
 3. IF account_balance >= purchase_amount THEN
 4. APPROVE payment
 5. ELSE
 6. REJECT payment
 7. END IF



We need
conditions

Lecture 5 - Boolean Algebra

- **George Boole (1815-1864)** - English mathematician
- developed a mathematical language for logic arguments
- Boolean algebra works with two values:
 - True
 - False
- It defines logical operations such as:
 - AND
 - OR
 - NOT
- Boolean algebra provides the mathematical foundation for expressing and combining conditions in a program.



Lecture 5 - Boolean Algebra

- A condition is an expression that can be evaluated as:
 - True
 - False
- Python provides `bool` type with possible values `True` and `False`
- They are used to represent the result of logical conditions
- They can also be stored in variables:

```
truth = 100 > 10
print(truth)

is_open = False
```

```
balance = 100
amount = 40

balance >= amount
# True
```

Lecture 5 - Boolean Algebra

- In Python, comparison operators producing Boolean values are:

```
x == y    # equal to
x != y    # not equal to
x < y     # less than
x <= y    # less than or equal to
x > y     # greater than
x >= y    # greater than or equal to
```

```
# Examples
```

```
age >= 18
```

```
price < budget
```

```
password == "python"
```

Lecture 5 - Boolean Algebra

- In Python, comparison operators producing Boolean values are:

```
x == y    # equal to
x != y    # not equal to
x < y     # less than
x <= y    # less than or equal to
x > y     # greater than
x >= y    # greater than or equal to
```

Examples

```
age >= 18
```

```
price < budget
```

```
password == "python"
```

NOTE: remember the difference:

- = → is assignment of a value
- == → is the comparison operator

Lecture 5 - Boolean Algebra

- Multiple conditions: boolean expressions can be combined with **logical operators**:
 - and
 - or
 - not

```
age >= 18 and has_ticket
```

```
is_student or age < 26
```

```
is_open = not is_closed
```

```
has_ticket and not is_closed
```

 Python tends to be
similar to natural
language

Lecture 5 - Boolean Algebra

- Logical Operators – Truth Tables

AND

x	y	x AND y
False	False	False
False	True	False
True	False	False
True	True	True

AND is True only if both operands are True

OR

x	y	x AND y
False	False	False
False	True	True
True	False	True
True	True	True

OR is True if at least one operands is True

NOT

x	NOT x
False	True
True	False

NOT reverses the Boolean value.

Lecture 5 - Boolean Algebra

- **Logical Operators – Properties**
 - **Commutative:**
 - $x \text{ AND } y == y \text{ AND } x$
 - $x \text{ OR } y == y \text{ OR } x$
 - **Associative:**
 - $(x \text{ AND } y) \text{ AND } z == x \text{ AND } (y \text{ AND } z)$
 - $(x \text{ OR } y) \text{ OR } z == x \text{ OR } (y \text{ OR } z)$
 - **Distributive:**
 - $x \text{ AND } (y \text{ OR } z) == (x \text{ AND } y) \text{ OR } (x \text{ AND } z)$
 - $x \text{ OR } (y \text{ AND } z) == (x \text{ OR } y) \text{ AND } (x \text{ OR } z)$

Lecture 5 - Boolean Algebra

- Logical Operators – Properties

- De Morgan's laws

- $\text{NOT } (x \text{ AND } y) == (\text{NOT } x) \text{ OR } (\text{NOT } y)$
 - $\text{NOT } (x \text{ OR } y) == (\text{NOT } x) \text{ AND } (\text{NOT } y)$

- Example: house is not secure if door is not locked and windows are not closed

- $\text{NOT } (\text{door_locked AND windows_closed}) \rightarrow \text{house is not secure}$
 - $(\text{NOT door_locked}) \text{ OR } (\text{NOT window_closed}) \rightarrow \text{house is not secure}$

Lecture 5 - Conditional Statement

- The if statement

- Use `if cond :` to execute code only when a condition is True

```
balance = 100  
amount = 40
```

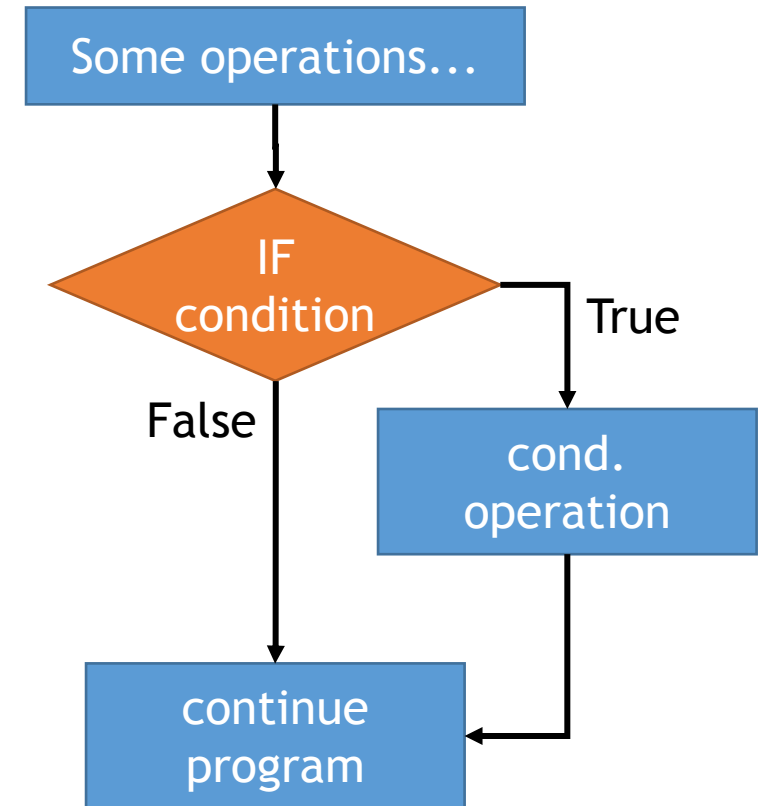
```
if balance >= amount:  
    print("Payment approved")
```

header

statement

block

compound
statement



Lecture 5 - Conditional Statement

- The `if` statement
 - Use `if cond :` to execute code only when a condition is True

```
balance = 100
amount = 40

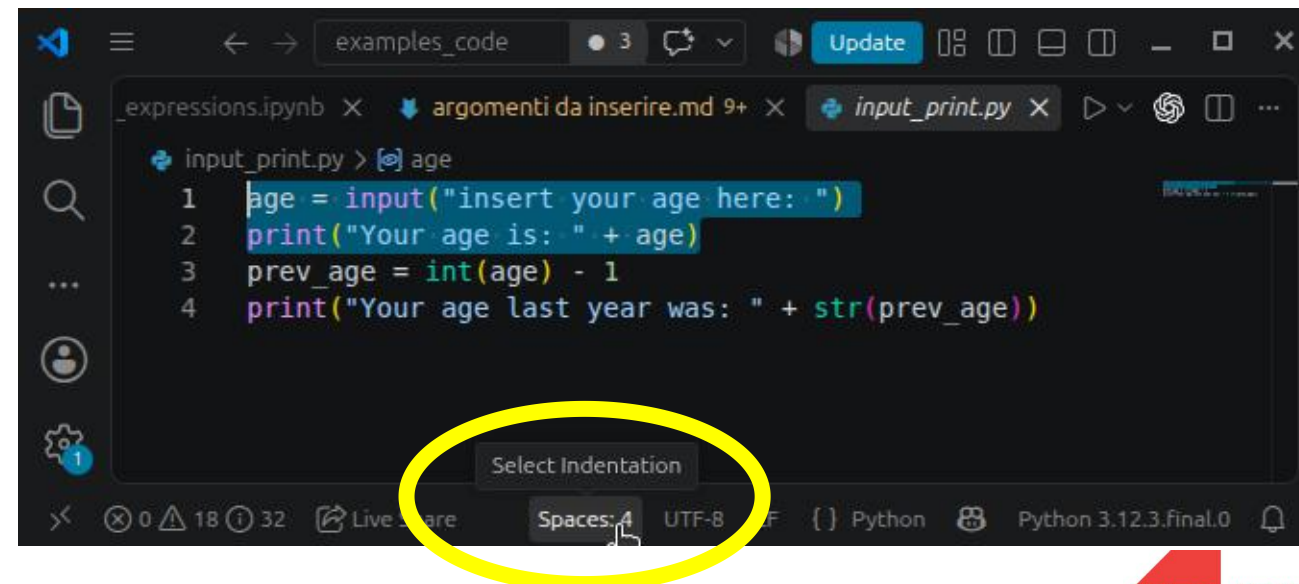
if balance >= amount:
    print("Payment approved")
```

Python uses **indented blocks** to identify the code executed only if the condition is True (statement block)

Lecture 5 - Conditional Statement

- **Details about indentation**
 - In Python, indentation is part of the **syntax**
 - Outside the block, the program continues
- How is it made? Spaces or Tabs?
 - Python allows both **spaces** and **tabs** for indentation.
 - Do not mix them in the same code: inconsistent indentation can also produce errors or unexpected block structure
 - The standard convention is **4 spaces** per indentation level → in editors, managed automatically (use tab, they put 4 spaces)

```
if condition:  
    indented block  
    indented block  
  
outside the block
```

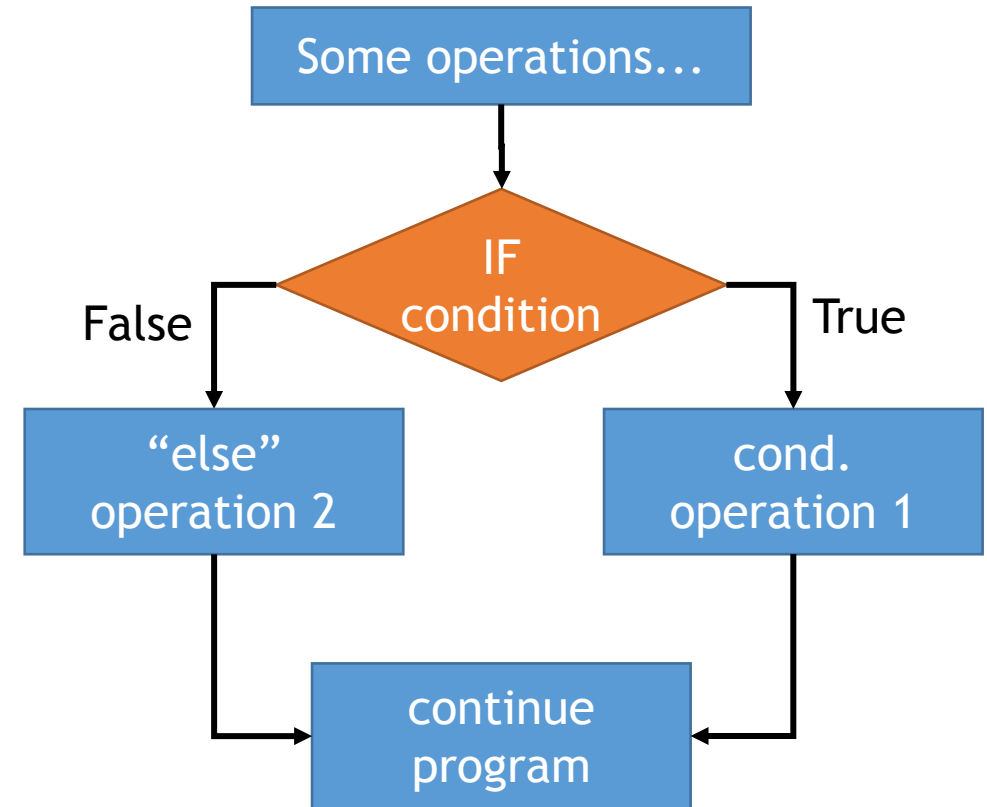


Lecture 5 - Conditional Statement

- We may have 2 alternatives:
 - If True, then do something
 - If False, then do something else

→ else: to manage any other case

```
if balance >= amount:  
    print("Payment approved")  
else:  
    print("Payment rejected")
```

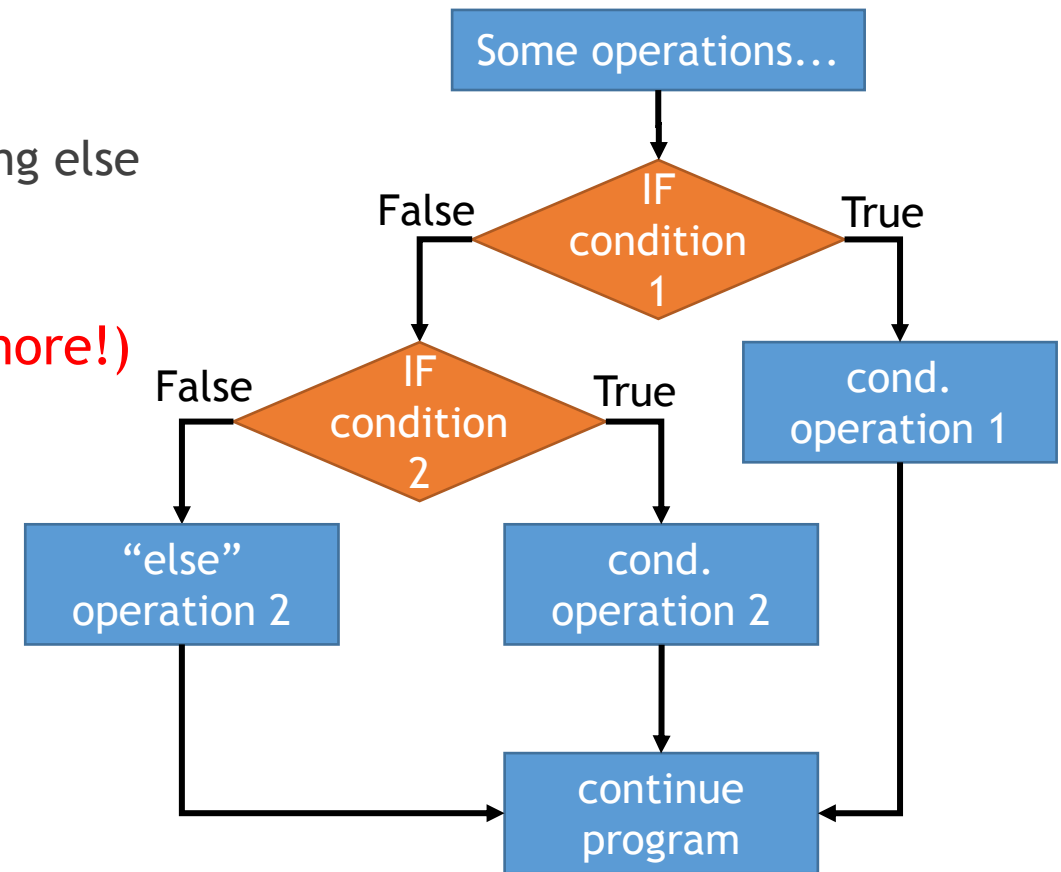


Lecture 5 - Conditional Statement

- We may have more than 2 alternatives:
 - If Expression 1 is True, then do something
 - Otherwise, if Expression 2 is True, then do something else
 - If both are False, then do something else again

→ `elif expression2` to manage another (one or more!) specific case (given the first is false)

```
if score >= 18:  
    print("Passed")  
elif score >= 15:  
    print("Almost passed")  
else:  
    print("Failed")
```

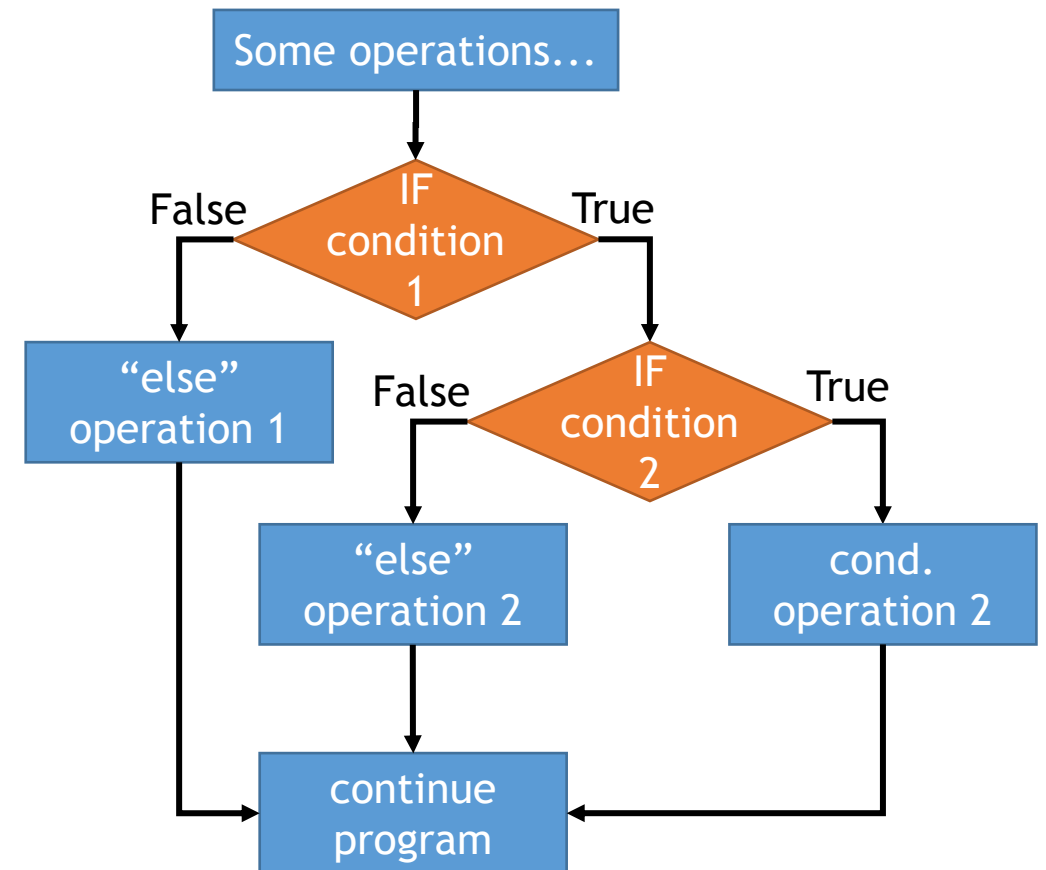


Lecture 5 - Conditional Statement

- **Nested conditions**

- An if statement can appear inside another if
- Nested conditions are useful when one decision depends on another

```
if age >= 18:  
    if has_ticket:  
        print("Access allowed")  
    else:  
        print("Ticket required")  
else:  
    print("Access denied")
```



Lecture 5 - Conditional Statement

- Comparing strings

- Strings can use the usual comparison operators:
- `==` `!=` `<` `<=` `>` `>=`
- Python compares strings **lexicographically**, character by character
- **Uppercase** letters come **before** lowercase letters in Unicode/ASCII ordering (“A” < “a”, “Z” < “a”)
- The comparison stops at the first different character.

```
"cat" == "cat"      # True
"apple" < "banana"  # True
"apple" < "apricot" # True
"Cat" < "cat"       # True
```

String comparison is based on **character codes** (UNICODE), not on "dictionary order"
→ Accented characters may not follow the expected alphabetical order

Lecture 5 - Conditional Statement

String comparison: important details

- Uppercase and lowercase characters have different Unicode code points
- Numeric-looking strings are still compared as strings. in ASCII/Unicode numbers are before letters
- Comparison is case-sensitive.

```
"Z" < "a"           # True
ord("Z")            # 90
ord("a")             # 97

"10" < "2"           # True
"2" < "A"            # True
```

If you want a numeric comparison,
convert first:

```
int("10") < int("2") # False
```

Lecture 5 - Conditional Statement

Boolean interpretation of values

- Python can interpret many values as True or False.
 - False values seen so far:
 - 0, 0.0, empty string ""
 - Non-zero numbers and non-empty strings are True.
- This implicit interpretation is often called *truthiness*.

```
bool(0)           # False
bool('0')         # True
bool(1)           # True
bool(-3)          # True
bool("")          # False
bool("hello")     # True
bool("False")     # True
```

Note: "0" is a non empty string, so it is "True". Python does not check the content of the string

Lecture 5 - Conditional Statement

Boolean interpretation of values

- Python can interpret many values as True or False.
 - False values seen so far:
 - 0, 0.0, empty string ""
 - Non-zero numbers and non-empty strings are True.
- This implicit interpretation is often called *truthiness*.

To check if a string contains only numbers, (so that it can be converted to `int()`), use method: `text.isdigit()`

```
bool(0)           # False
bool('0')         # True
bool(1)           # True
bool(-3)          # True
bool("")          # False
bool("hello")     # True
bool("False")     # True
```

```
if age_text.isdigit():
    age = int(age_text)
```

Lecture 5 - Conditional Statement

Truthiness directly inside an if

- The condition of an if does not need to be explicitly transformed into a boolean object using `bool()`
- Python automatically evaluates the truthiness of the value

```
name = input("Name: ")
if name:
    print("Name entered")
else:
    print("Empty input")

# Equivalent idea
if bool(name):
    ...
```

Hands-on!

Lecture 5 - Exercises

- Conditional statements in practice
 - Guided exercise: validate user input before making a decision
 - Laboratory exercise: translate a real decision process into code
 - Main idea: correct rules are not enough if the input is not controlled

Conditions help us both choose an action and protect the program from invalid data.

Lecture 5 - Guided Exercise

Ex. 1: Ticket price and input validation

- A museum asks the visitor to enter their age.
- Rules:
 - Under 18 → reduced ticket (€7)
 - 18 or older → full-price ticket (€12)
 - Accepted ages: 0 to 120
- But `input()` always returns a string.

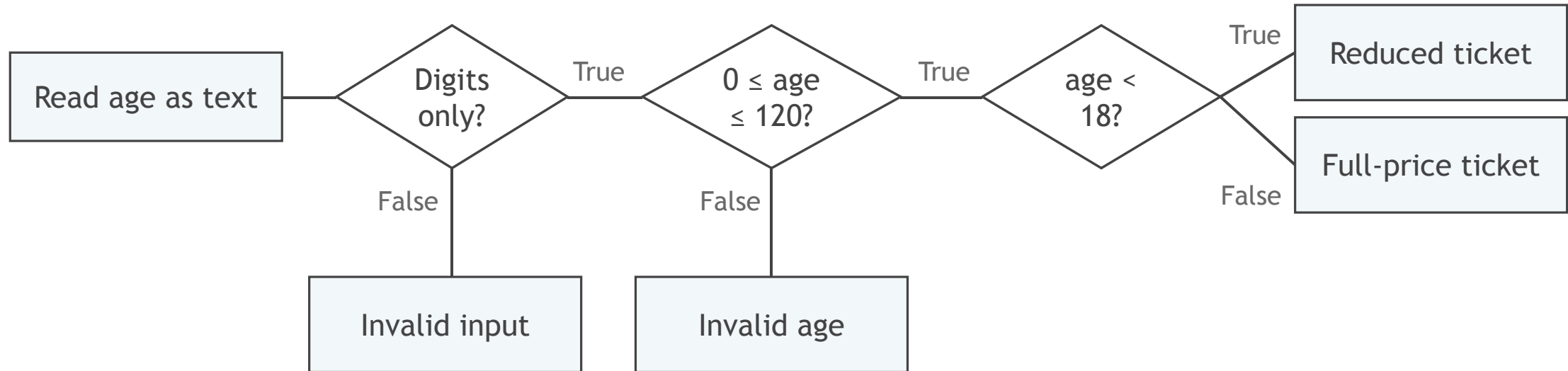
```
age = int(input("Enter your age: "))
```

What happens if the user enters "hello", "-3", or "150"?

Should we convert first... or validate first?

Lecture 5 - Guided Exercise

A safer decision process



Validation happens before the program relies on the value.

Lecture 5 - Guided Exercise

Some hints

- create a new file: ticket.py
- Complete the three missing conditions:
 - 1. Is the input made only of digits?
 - 2. Is the age outside the accepted range?
 - 3. Is the visitor eligible for the reduced ticket?
- Then test both valid and invalid inputs.

```
age_text = input("Enter your age: ").strip()

if _____:
    age = int(age_text)

    if _____:
        print("Invalid age")
    elif _____:
        print("Reduced ticket")
    else:
        print("Full-price ticket")
else:
    print("Invalid input")
```

Lecture 5 - Guided Exercise

Some hints

- create a new file: ticket.py
- Complete the three missing conditions:
 - 1. Is the input made only of digits?
 - 2. Is the age outside the accepted range?
 - 3. Is the visitor eligible for the reduced ticket?
- Then test both valid and invalid inputs.

`text.strip()` is a method that “cleans” a string from possible spaces at the beginning or ending:

" Hello! ".strip() --> "Hello!"
Here it is applied directly to the `input()` result

```
age_text = input("Enter your age: ").strip()

if _____:
    age = int(age_text)

    if _____:
        print("Invalid age")
    elif _____:
        print("Reduced ticket")
    else:
        print("Full-price ticket")
else:
    print("Invalid input")
```

Lecture 5 - Laboratory Exercise

Ex. 2: Simplified income tax calculator

- Write a program that calculates income tax using a simplified tax schedule.
- Two pieces of information are required:
 - annual income
 - marital status: Single or Married
- Tax rates:
 - 10% up to the applicable threshold
 - 25% on the part above the threshold

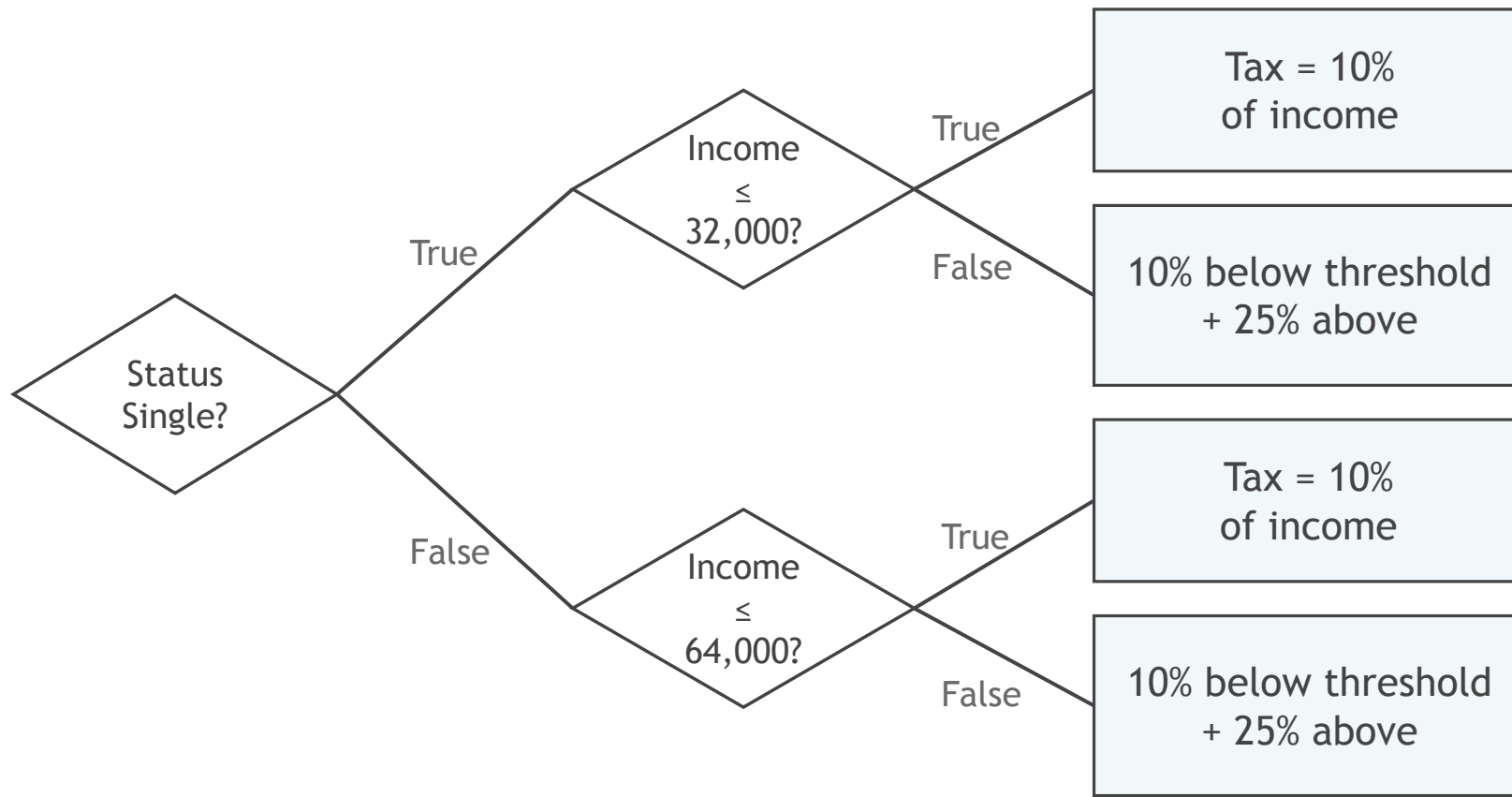
Single
threshold:
€32,000

Married
threshold:
€64,000

The same tax rates apply, but the
threshold changes.

Lecture 5 - Laboratory Exercise

Hints: Four decision branches



**Before coding:
identify the
conditions,
constants and four
possible paths.**

Lecture 5 - Laboratory Exercise

Do not trust the input

- Your calculation may be correct and still produce a bad program.
- Consider these user inputs:
 - `income = "hello"`
 - `income = "-5000"`
 - `income = " 5000"`
 - `status = "MARRIED"`
 - `status = " married "`
 - `status = "divorced"`
- Decide what your program should **accept**, **normalize** or **reject**.

Input validation is part of the algorithm.

Minimum requirements:

- validate income before numeric conversion
- reject negative or non-numeric income
- accept only the expected marital-status values
- print a meaningful error message

Good practice:

- use `.lower()/.upper()` and `.strip()` to normalize strings
- check numeric data with `.isdigit()`

“It is impossible to
make anything
foolproof, because
fools are so ingenious.”



Anonymous, variant of Murphy's Law

Lecture 5 - Laboratory Exercise

Implementation and testing

- Build the program from scratch.
- Suggested steps:
 - 1. Define rates and thresholds as constants
 - 2. Read and validate the two inputs
 - 3. Select the correct threshold
 - 4. Compute the tax
 - 5. Print the result clearly.

Test cases

30000, S	→	€3,000
40000, S	→	€5,200
60000, M	→	€6,000
70000, M	→	€8,900
hello, S	→	invalid
-5000, M	→	invalid
40000, X	→	invalid

A program is not finished until you have tested its different branches.

Lecture 5 - part 2

built-in functions and modules

Lecture 5 - built-in functions, modules, and floats

- From conditions to useful tools
 - basic mathematical built-in functions
 - libraries, modules and import styles
 - the `math` module
 - floating-point comparisons and `isclose()`

Goal: write clearer programs by using existing functions correctly.

```
abs(-7)
round(3.1416, 2)
min(4, 9, 2)
max(4, 9, 2)

import math
math.sqrt(2)
```

Lecture 5 - Basic Mathematical Functions

Introducing some new functions

- Python provides useful built-in functions for simple numerical operations:
- `abs(x)` → absolute value
- `round(x)` → rounded value
- `round(x, n)` → round to n decimal places
- `min(...)` → smallest value
- `max(...)` → largest value

```
abs(-7)           # 7
round(3.1416)      # 3
round(3.1416, 2)   # 3.14
min(4, 9, 2)       # 2
max(4, 9, 2)       # 9
```

These are built-in functions, not methods.

Full list of built-in functions: <https://docs.python.org/3/library/functions.html>

Lecture 5 - Basic Mathematical Functions

Using `round()` in output and calculations

- `round()` can be used with one or two arguments.
 - `round(x)` returns the nearest integer.
 - `round(x, n)` returns `x` rounded to `n` decimal places.
- Very useful for prices, rates and percentages.
- Difference with `int()`:
 - `int(N)` truncates the decimal part
 - `round(N)` rounds to the nearest integer.

```
price = 12.567
tax = 2.513

total = price + tax

round(total, 2)          # 15.08
print(f"€{total:.2f}")  # €15.08

int(3.9)                 # 3
round(3.9)               # 4
```

For formatted printing, f-strings are often clearer than changing the value itself.

Lecture 5 - Libraries and Modules

- A **module** is a file containing Python code that can be reused.
- A library is a collection of modules and tools.
- Why use them?
 - avoid rewriting existing code
 - use reliable and tested functions
 - organize programs into reusable parts

```
# use code written elsewhere
import math

math.sqrt(16)
```

Lecture 5 - Ways of Importing

- There are different ways to access a module.
 - `import module_name`
 - call functions with dot notation
`module_name.func()`
 - `from module_name import func1, ...`
 - `from module_name import *` (all)
 - call functions directly `func()`
- Recommended for beginners:
 - import the module and use dot notation
- This keeps the origin of each function clear.

```
# 1. Import the module
import math

x = math.sqrt(25) # 5.0
p = x * math.pi  # 5 * 3.1415...
```

```
# 2. Import one function
from math import sqrt, pi

x = sqrt(25)      # 5.0
p = x * pi        # 5 * 3.1415...
```

```
# 3. Alias
import math as m

x = m.sqrt(25) # 5.0
p = x * m.pi  # 5 * 3.1415...
```

Lecture 5 - The math Module

- The math module provides mathematical functions and constants.
- It is part of the *Python Standard Library*
- It must be imported before use
- Examples of math functions and constants:
 - `pi` → the mathematical constant π
 - `sqrt()` → Square root of a value
 - `ceil()` → returns the smallest int $\geq$
 - `floor()` → returns the greatest int $\leq$

```
import math

math.sqrt(16)      # 4.0
math.sqrt(2)       # 1.414...

math.pi           #  $\pi$ 
math.ceil(3.2)     # 4
math.floor(3.8)    # 3
math.floor(-3.8)   # -4
```

`sqrt(x)` returns a float, even when the result is mathematically an integer.

docs of math module: <https://docs.python.org/3/library/math.html>

Lecture 5 - Floating-Point Precision

- Due to the limited precision of binary representation, some decimal or irrational values **cannot be represented exactly**.
- Operations may produce very small rounding errors.
- Therefore, direct **equality between floats** can be unsafe.

```
import math

r = math.sqrt(2)
y = r ** 2

print(y)          # 2.000000000000000004
print(y == 2)     # False
```

Mathematically: $(\sqrt{2})^2 = 2$
Computationally: the stored value may be only approximate.

Lecture 5 - Comparing Floats with `isclose()`

- When comparing floats, ask whether two values are close enough (see Lect. 3).
- `math.isclose(a, b)` returns `True` when *a* and *b* are sufficiently close.
- This is usually better than using `==` with computed floats.
- It can be specified as optional parameter the relative or the absolute tolerance.

```
import math

y = math.sqrt(2) ** 2

print(y == 2)
# False

print(math.isclose(y, 2))
# True
```

Use `==` for integers and exact values.
Be careful with `==` after floating-point calculations.

Lecture 5 - Python Libraries

Why Python is so popular

- One of Python's major strengths is its large ecosystem of **modules** and **libraries**.
- Many common tasks can be solved without writing everything from scratch.
- Libraries are available for:
 - mathematics and statistics;
 - data analysis and visualization;
 - web development;
 - machine learning;
 - automation;
 - scientific computing;
 - many other application areas.

We cannot cover all the libraries, there are many specific courses specifically built for each of them

Lecture 5 - Python Libraries

The Python Standard Library: “batteries included”

- Python already comes with a rich Standard Library.
 - Examples:
 - `math` → mathematical functions
 - `random` → random numbers
 - `statistics` → basic statistics
 - `datetime` → dates and times
 - `csv` → CSV files
 - `os` / `pathlib` → files and directories
- No extra installation is required.

Lecture 5 - Python Libraries

External libraries

- Many useful libraries are not part of the Standard Library.
- Examples:
 - NumPy → arrays and numerical computing
 - pandas → data tables and analysis
 - matplotlib → plots and charts
 - SciPy → scientific computing tools
 - requests → web requests and APIs
 - scikit-learn → machine learning tools
- They must normally be installed separately.

Lecture 5 - Python Libraries

Installing a library

- Python, depending on the installation, provides different managers.
- The most common is **pip**
 - Example installation in your terminal running:
 - `python -m pip install numpy`
 - `python -m pip install pandas`
 - `python -m pip install matplotlib`
 - Then, the modules can be imported as usual:
 - `import numpy as np`
 - `import pandas as pd`
 - `import matplotlib as plt`

NOTE: Modules and packages can be organized hierarchically: a package may contain subpackages and submodules.

```
import matplotlib.pyplot
matplotlib.pyplot.plot([1, 2, 3])
```

```
from matplotlib.pyplot import plot
plot([1, 2, 3])
```

Exercises

Lab + some problems for independent training

Lecture 5 - Laboratory Activity

Delivery cost estimator

- A small shop wants to estimate the delivery cost for an online order.
- The program asks for:
 - order amount
 - distance from the shop
 - customer type: R = regular, P = premium
- The program must validate the input before calculating the cost.

Business rules

Premium customer:
free delivery if order $\geq$ €50

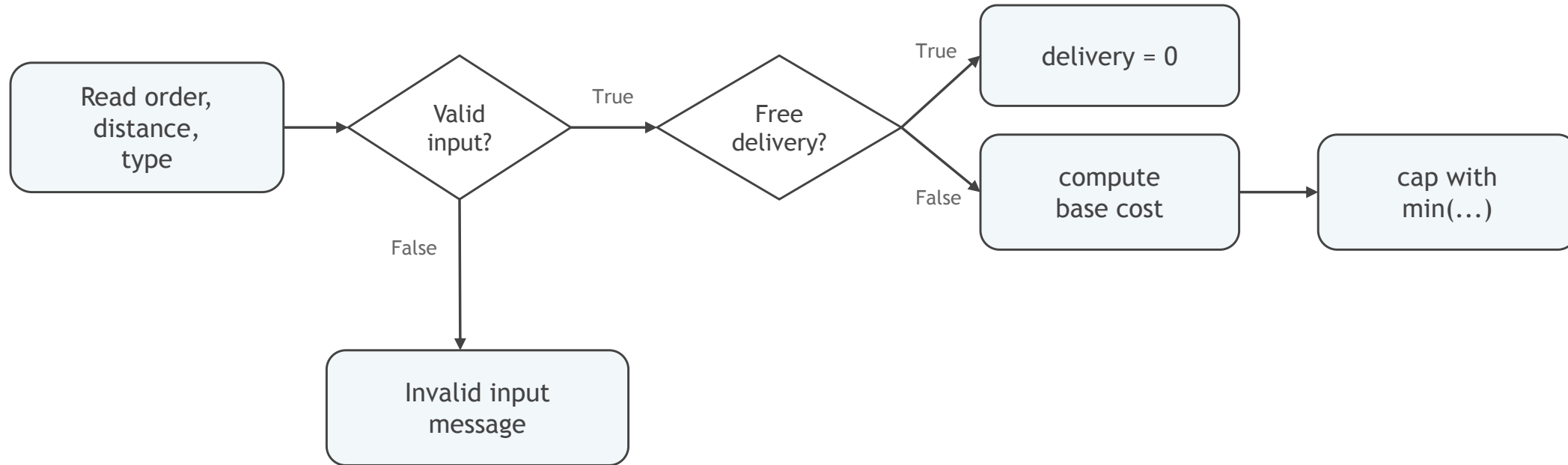
Regular customer:
free delivery if order $\geq$ €80

Otherwise:
delivery = €4 + €0.50 per km

Maximum delivery charge:
€15

Lecture 5 - Laboratory Activity

Decision structure



Use conditions to separate validation, decision and calculation.

Lecture 5 - Laboratory Activity

Implementation hints

- Suggested implementation steps:
 - 1. Read the three inputs as strings.
 - 2. Normalize the customer type with `.strip().upper()`.
 - 3. Validate numeric inputs before converting.
 - 4. Use `if / elif / else` to check free delivery.
 - 5. Use `min()` to apply the maximum charge.
 - 6. Use an f-string to print the result.

```
order_text    = input("Order amount: ")
distance_text = input("Distance in km: ")
customer_type = input("Customer type [R/P]: ")

customer_type = customer_type.strip().upper()

# Then validate, convert and decide...
```

Test also invalid inputs: negative numbers, text instead of numbers, unknown customer type.

Independent Exercise 1

Right-triangle checker

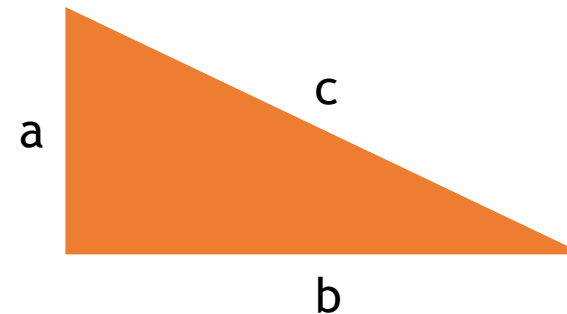
- Ask the user for the lengths of three sides: a, b, c.
- Assume c should be the longest side.
- Your program must:
 - validate that all values are positive numbers;
 - check whether c is really the longest side;
 - use `math.sqrt()` to compute the expected hypotenuse;
 - use `math.isclose()` to compare floating-point values;
 - print a clear final message.

```
import math

expected_c = math.sqrt(a ** 2 + b ** 2)

if math.isclose(expected_c, c):
    print("Right triangle")
```

Do not compare floats with `==` when small rounding errors are possible.



Independent Exercise 2

Grade decision assistant

- Ask the user for:
 - test grade, from 0 to 20;
 - project grade, from 0 to 20;
 - project submitted? Y/N.
- The final grade is:
 - 70% test + 30% project
- The student passes only if:
 - test grade ≥ 7
 - final grade ≥ 10
 - the project was submitted

```
final_grade = 0.70 * test + 0.30 * project  
final_grade = round(final_grade, 1)
```

```
# Then decide:  
# passed / failed  
# and explain why
```

**Use Boolean expressions with and / or
to make the decision readable.**

Independent Exercise 3

Price correction and comparison

- A company regularly purchases a product from a supplier. The supplier proposes a new unit price. The manager accepts any price reduction, but a price increase is accepted only if it does not exceed a given maximum amount.
- Ask for:
 - old price;
 - new price;
 - maximum accepted increase in euros.
- Your program must:
 - validate that prices are positive;
 - compute the absolute price difference with `abs()`;
 - identify the lower and higher price with `min()` and `max()`;
 - use `round(..., 2)` for printed monetary values;
 - accept the new price if it is lower than the old price or if the increase does not exceed the accepted threshold.

```
difference = abs(new_price - old_price)

lower = min(old_price, new_price)
higher = max(old_price, new_price)

if new_price <= old_price:
    ...
if difference <= limit:
    print("Change accepted")
```

A small numerical function can make a condition easier to express.

Lecture 5 - Practice Checklist

- Before finishing, verify that your programs:
 - read inputs as strings;
 - validate inputs before using them;
 - convert values only when safe;
 - use conditions to separate cases;
 - print meaningful messages for invalid data;
 - use built-in functions to simplify calculations;
 - avoid unsafe float comparisons when needed.

**A correct formula is not enough.
A good program must also handle bad input.**

Test each decision branch.