UNI FREIBURG

Python Course

WT 25/26

Lecture: Marcus Rockel & Prof. Dr. Eva Lütkebohmert-Holtz, Tutorial: Felix Häschel
https://www.finance.uni-freiburg.de/

Tutorial 2

Exercise 1 (Boolean Logic)**.**
Evaluate the following boolean expression **without running any code**. Write down the final result (`True` or `False`) and then verify your answer by running the code to see if you were correct.

```
result = (True or False) and (not True or False) and (True or not False) or (not (True and
↪  False) and (True or False)) and (True and False or not False)
```

Optional: Truthy and Falsy Values
Evaluate the following expression using the predefined variables. Some values in Python are considered *truthy* (treated as `True`) or *falsy* (treated as `False`) in boolean contexts. Determine each variable's boolean equivalent, calculate the final result, and then verify it by running the code.

```
A = 0
B = "hello"
C = []
D = [1, 2]
E = ""
F = -3
G = {}

result_optional = ((A and not B) or (C and not D)) and (not E or (F and G)) or (not (A or
↪  C) and (D or not F)) and (E or not G)
```

In Python, logical operators are processed in the following order:

(i) `not` (highest priority)

(ii) `and`

(iii) `or` (lowest priority)

Use parentheses `()` to override the default order when needed.

Solution.

(i) The first expression evaluates to `True`.

(ii) For the optional part:
The boolean equivalents are:

$$A : \texttt{False}, \quad B : \texttt{True}, \quad C : \texttt{False}, \quad D : \texttt{True},$$
$$E : \texttt{False}, \quad F : \texttt{True}, \quad G : \texttt{False}.$$

Substituting and simplifying yields `True`.

Exercise 2 (Set Operations)**.**
In Python, the *symmetric difference* between two sets A and B includes elements that are in either A or B, but not in both. The symmetric difference can be computed directly with `A.symmetric_difference(B)` or `A ^ B`.

Replicate this operation using the other set operations: `union (|)`, `intersection (&)`, and `difference (-)`. Use the provided sets to verify your result.

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Symmetric Difference
print(f'Symmetric Difference: {A ^ B}')
```

Solution.

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# Symmetric Difference
print(f'Symmetric Difference: {A ^ B}')

# Solution 1
sol_1 = (A | B) - (A & B)
print(f'Solution 1: {sol_1}')

# Solution 2
sol_2 = (A - B) | (B - A)
print(f'Solution 2: {sol_2}')
```

(i) Symmetric Difference: {1,2,5,6}

(ii) Both alternative solutions produce the same result: {1,2,5,6}

Exercise 3 (Password Strength Check)**.**
A password is considered *strong* if it meets the following criteria:

- At least 8 characters long,
- Contains at least one uppercase letter,
- Contains at least one digit,
- Contains at least one special character that is not a letter or digit.

Write code to check if a given password meets these criteria, copy `Python2025` as a string into the Python shell and evaluate its strength.
Hint: use `set` and membership.

Solution. The expression evaluates to `Strong` for the given password.

```
# 1. Define the password from the problem description
pwd = "P thon2025"

# 2. Define the character sets
uppercase_chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
UPPERS = set(uppercase_chars)
LOWERS = set(uppercase_chars.lower())
```

```
DIGITS = set("0123456789")

# 3. Check the criteria
has_len = (len(pwd) >= 8)
has_upper = bool(set(pwd) & UPPERS)
has_digit = bool(set(pwd) & DIGITS)

# 4. Check for special char (anything not Latin alphanumeric)
# The problem defines "special" as "not a letter or digit".
has_special = bool(set(pwd) - (UPPERS | LOWERS | DIGITS))

# 5. Determine result
strong = has_len and has_upper and has_digit and has_special

print(f"Password '{pwd}' is {'Strong' if strong else 'Weak'}")
```

Exercise 4 (Leap Year Check)**.**
The concept of leap years was introduced by **Julius Caesar** in 45 BCE with the **Julian calendar**, which added one extra day every four years. In 1582, the **Gregorian calendar** refined this system to correct the small yearly drift caused by the Julian rule.

A year is a leap year if it is divisible by 4, except for years divisible by 100. However, if a year is divisible by both 100 and 400, it is still a leap year. For example, 2000 was a leap year because it is divisible by 400, whereas 1900 was not.

In this exercise, we follow the modern (Gregorian) rules but allow checking back until Caesar's introduction of leap years. Years before 45 BCE should not be accepted.

- Ask the user to enter a year using `input()`.
- Apply conditional statements to determine whether the year is a leap year according to the modern rules.
- Print an appropriate message to the console depending on the result (e.g., `"2024 is a leap year."` or `"2023 is not a leap year."`).
- If the entered year is before 45 BCE, print a message indicating that such years are not valid for this check.

Solution.

```
# Leap Year Checker

# Ask the user for input
year = int(input("Enter a year: "))

# Check for years before the introduction of leap years
if year < -44:  # 45 BCE is represented as -44 in astronomical year numbering
    print("Please enter a year from 45 BCE onward (Julian calendar introduced by
    ↪  Caesar).")

# Check according to Gregorian leap year rules
elif (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

Exercise 5 (Secret Word - String Operations)**.**
You have been given a long string of words without any particular order or meaning. The words together form a continuous text stored in the variable `words`.

```
words = (
    "  i a to a "
    "people write code in school index split join slice print tests memory system "
    "design object number string module import kernel tree proof result level order "
    "focus topic figure work skill theory beyond "
    "a is a note once help test yeti pro "
)
```

Your task is to carefully follow the instructions below to transform this text step by step and extract a hidden secret word.

- Remove all whitespace and spaces from `words` and replace every occurrence of the letter `'x'` with `'p'`.
- Split the resulting string on the letter `'a'` to obtain a list.
- From this list, take the first two and the last two elements and join them together without a separator.
- Slice the resulting string using `[-3:1:-4]` to obtain the secret word.

Solution.

```
# Given list of words
words = (
    "  i a to a "
    "people write code in school index split join slice print tests memory system "
    "design object number string module import kernel tree proof result level order "
    "focus topic figure work skill theory beyond "
    "a is a note once help test yeti pro "
)

# Clean text: strip whitespace, replace letters, remove spaces
words = words.strip()
words = words.replace("x", "p")
words = words.replace(" ", "")

# Split text and recombine parts
parts = words.split("a")
joined = "".join(parts[:2] + parts[-2:])

# Slice backwards to extract the secret word
secret = joined[-3:1:-4]

print(secret)
```

Exercise 6 (Visual Studio Code installation)**.**
Install Visual Studio Code from Visual Studio Code on your computer. Then, open VS Code and install the Python extension from the Extensions view (`Ctrl + Shift + X`). Create a new Python file `hello_world.py`, write a simple "Hello World" program and run it in Visual Studio Code.

Note: Consult the VS Code Tutorial for Python, if questions or problems arise. You do **not** need to create a virtual environment.

Exercise 7 (Jupyter Notebook installation)**.**
Open VS Code and install the Jupyter extension from the Extensions view. In the terminal, run the following to install the required packages:

```
pip install jupyter
```

Create a new Jupyter Notebook `hello_world.ipynb`, write another "Hello World" program and run it in the Jupyter Notebook.

Note: Consult the Jupyter Notebook documentation or this introductory tutorial on Real Python, if questions or problems arise.