LAB 1: Print

Estimated time

5-10 minutes

Level of difficulty

Very easy

Objectives

- becoming familiar with the `print()` function and its formatting capabilities;
- experimenting with Python code.

Scenario

The `print()` command, which is one of the easiest directives in Python, simply prints out a line to the screen.

In your first lab:

- use the `print()` function to print the line `Hello, Python!` to the screen. Use double quotes around the string;
- having done that, use the `print()` function again, but this time print your first name;
- remove the double quotes and run your code. Watch Python's reaction. What kind of error is thrown?
- then, remove the parentheses, put back the double quotes, and run your code again. What kind of error is thrown this time?
- experiment as much as you can. Change double quotes to single quotes, use multiple `print()` functions on the same line, and then on different lines. See what happens.

LAB 2: Literal

Estimated time

5-10 minutes

Level of difficulty

Easy

Objectives

- becoming familiar with the `print()` function and its formatting capabilities;
- practicing coding strings;
- experimenting with Python code.

Scenario

Write a one-line piece of code, using the `print()` function, as well as the newline and escape characters, to match the expected result outputted on three lines.

Expected output

```
"I'm"

""learning""

"""Python"""
```

LAB 3: Variables

Estimated time

10 minutes

Level of difficulty

Easy

Objectives

- becoming familiar with the concept of storing and working with different data types in Python;
- experimenting with Python code.

Scenario

Here is a short story:

Once upon a time in Appleland, John had three apples, Mary had five apples, and Adam had six apples. They were all very happy and lived for a long time. End of story.

Your task is to:

- create the variables: `john`, `mary`, and `adam`;
- assign values to the variables. The values must be equal to the numbers of fruit possessed by John, Mary, and Adam respectively;
- having stored the numbers in the variables, print the variables on one line, and separate each of them with a comma;
- now create a new variable named `total_apples` equal to addition of the three former variables.
- print the value stored in `total_apples` to the console;

experiment with your code: create new variables, assign different values to them, and perform various arithmetic operations on them (e.g., +, -, *, /, //, etc.). Try to print a string and an integer together on one line, e.g., `"Total number of apples:"` and `total_apples`.

LAB 4: Variable & Operator

Estimated time

10-15 minutes

Level of difficulty

Easy

Objectives

- becoming familiar with the concept of numbers, operators, and arithmetic operations in Python;
- performing basic calculations.

Scenario

Take a look at the code in the editor: it reads a `float` value, puts it into a variable named `x`, and prints the value of a variable named `y`. Your task is to complete the code in order to evaluate the following expression:

$$3x^3 - 2x^2 + 3x - 1$$

The result should be assigned to `y`.

Remember that classical algebraic notation likes to omit the multiplication operator - you need to use it explicitly. Note how we change data type to make sure that `x` is of type `float`.

Keep your code clean and readable, and test it using the data we've provided, each time assigning it to the `x` variable (by hardcoding it). Don't be discouraged by any initial failures. Be persistent and inquisitive.

Test Data

Sample input

`x = 0`

`x = 1`

`x = -1`

Expected Output

`y = -1.0`

```
y = 3.0
```

```
y = -9.0
```

```
x =  # hardcode your test data here
x = float(x)
# write your code here
print("y =", y)
```

LAB 5: Input & Operator

Estimated time

5-10 minutes

Level of difficulty

Easy

Objectives

- becoming familiar with the inputting and outputting of data in Python;
- evaluating simple expressions.

Scenario

Your task is to complete the code in order to evaluate the results of four basic arithmetic operations.

The results have to be printed to the console.

You may not be able to protect the code from a user who wants to divide by zero. That's okay, don't worry about it for now.

Test your code - does it produce the results you expect?

We won't show you any test data - that would be too simple.

```
# input a float value for variable a here

# input a float value for variable b here


# output the result of addition here

# output the result of subtraction here

# output the result of multiplication here

# output the result of division here


print("\nThat's all, folks!")
```

LAB 6: Input

Estimated time

15-20 minutes

Level of difficulty

Easy

Objectives

- improving the ability to use numbers, operators, and arithmetic operations in Python;
- using the `print()` function's formatting capabilities;
- learning to express everyday-life phenomena in terms of programming language.

Scenario

Your task is to prepare a simple code able to evaluate the **end time** of a period of time, given as a number of minutes (it could be arbitrarily large). The start time is given as a pair of hours (0..23) and minutes (0..59). The result has to be printed to the console.

For example, if an event starts at **12:17** and lasts **59 minutes**, it will end at **13:16**.

Don't worry about any imperfections in your code - it's okay if it accepts an invalid time - the most important thing is that the code produce valid results for valid input data.

Test your code carefully. Hint: using the `%` operator may be the key to success.

1. Input Hour,Minute, Duration
2. Calculate Total Minutes
3. Add duration of Total Minutes
4. Calculate Hour and minute of the end time

Test Data

Sample input:`12`

`17`

`59`

Expected output: `13:16`

Sample input:`23`

`58`

`642`

Expected output: `10:40`

Sample input:`0`

`1`

`2939`

Expected output: `1:0`

LAB 7: Comparison

Estimated time

5-10 minutes

Level of difficulty

Very Easy

Objectives

- becoming familiar with the `input()` function;
- becoming familiar with comparison operators in Python.

Scenario

Using one of the comparison operators in Python, write a simple two-line program that takes the parameter `n` as input, which is an integer, and prints `False` if `n` is less than `100`, and `True` if `n` is greater than or equal to `100`.

Don't create any `if` blocks (we're going to talk about them very soon). Test your code using the data we've provided for you.

Test Data

Sample input: `55`

Expected output: `False`

Sample input: `99`

Expected output: `False`

Sample input: `100`

Expected output: `True`

Sample input: `101`

Expected output: `True`

Sample input: `-5`

Expected output: `False`

Sample input: `+123`

Expected output: `True`

LAB 8: Comparison Operator

Estimated time

5-15 minutes

Level of difficulty

Easy

Objectives

- becoming familiar with the `input()` function;
- becoming familiar with comparison operators in Python;
- becoming familiar with the concept of conditional execution.

Scenario

Spathiphyllum, more commonly known as a peace lily or white sail plant, is one of the most popular indoor houseplants that filters out harmful toxins from the air. Some of the toxins that it neutralizes include benzene, formaldehyde, and ammonia.

Imagine that your computer program loves these plants. Whenever it receives an input in the form of the word `Spathiphyllum`, it involuntarily shouts to the console the following string: `"Spathiphyllum is the best plant ever!"`

Write a program that utilizes the concept of conditional execution, takes a string as input, and:

- prints the sentence `"Yes - Spathiphyllum is the best plant ever!"` to the screen if the inputted string is `"Spathiphyllum"` (upper-case)
- prints `"No, I want a big Spathiphyllum!"` if the inputted string is `"spathiphyllum"` (lower-case)
- prints `"Spathiphyllum! Not [input]!"` otherwise. Note: `[input]` is the string taken as input.

Test your code using the data we've provided for you. And get yourself a Spathiphyllum, too!

LAB 9: Number Classification

Estimated time

5-10 minutes

Level of difficulty

Easy

Objectives

- Familiarize with the if-else instruction to branch the control path.
- Build a complete program that solves a simple real-life problem.

Scenario

Write a program that takes an integer input from the user and classifies it as positive, negative, or zero. The program should then print an appropriate message.

1. Use the input() function to take an integer input from the user.
2. Use an if-else instruction to check whether the number is positive, negative, or zero.
3. Print the appropriate message based on the classification.

LAB 10: Year Leap

Estimated time

10-25 minutes

Level of difficulty

Easy/Medium

Objectives

Familiarize the student with:

- using the `if-elif-else` statement;
- finding the proper implementation of verbally defined rules;
- testing code using sample input and output.

Scenario

As you surely know, due to some astronomical reasons, years may be *leap* or *common*. The former are 366 days long, while the latter are 365 days long.

Since the introduction of the Gregorian calendar (in 1582), the following rule is used to determine the kind of year:

- if the year number isn't divisible by four, it's a *common year*;
- otherwise, if the year number isn't divisible by 100, it's a *leap year*;
- otherwise, if the year number isn't divisible by 400, it's a *common year*;
- otherwise, it's a *leap year*.

Look at the code in the editor - it only reads a year number, and needs to be completed with the instructions implementing the test we've just described.

The code should output one of two possible messages, which are `Leap year` or `Common year`, depending on the value entered.

It would be good to verify if the entered year falls into the Gregorian era, and output a warning otherwise: `Not within the Gregorian calendar period`. Tip: use the `!=` and `%` operators.

Test your code using the data we've provided.

Test Data

Sample input: `2000`

Expected output: `Leap year`

Sample input: `2015`

Expected output: `Common year`

Sample input: `1999`

Expected output: `Common year`

Sample input: `1996`

Expected output: `Leap year`

Sample input: `1580`

Expected output: `Not within the Gregorian calendar period`

LAB 11: Secret Number

Estimated time

15 minutes

Level of difficulty

Easy

Objectives

Familiarize the student with:

- using the `while` loop;
- reflecting real-life situations in computer code.

Scenario

A junior magician has picked a secret number. He has hidden it in a variable named `secret_number`. He wants everyone who run his program to play the *Guess the secret number* game, and guess what number he has picked for them. Those who don't guess the number will be stuck in an endless loop forever! Unfortunately, he does not know how to complete the code.

Your task is to help the magician complete the code in the editor in such a way so that the code:

- will ask the user to enter an integer number;
- will use a `while` loop;
- will check whether the number entered by the user is the same as the number picked by the magician. If the number chosen by the user is different than the magician's secret number, the user should see the message `"Ha ha! You're stuck in my loop!"` and be prompted to enter a number again. If the number entered by the user matches the number picked by the magician, the number should be printed to the screen, and the magician should say the following words: `"Well done, muggle! You are free now."`

The magician is counting on you! Don't disappoint him.

EXTRA INFO

By the way, look at the `print()` function. The way we've used it here is called *multi-line printing*. You can use **triple quotes** to print strings on multiple lines in order to make text easier to read, or create a special text-based design. Experiment with it.

```
secret_number = 777


print(
"""
+================================+
| Welcome to my game, muggle!    |
| Enter an integer number        |
| and guess what number I've     |
| picked for you.                |
| So, what is the secret number? |
+================================+
""")
```

LAB 12: Counting

Estimated time

5-15 minutes

Level of difficulty

Very easy

Objectives

Familiarize the student with:

- using the `for` loop;
- reflecting real-life situations in computer code.

Scenario

Do you know what Mississippi is? Well, it's the name of one of the states and rivers in the United States. The Mississippi River is about 2,340 miles long, which makes it the second longest river in the United States (the longest being the Missouri River). It's so long that a single drop of water needs 90 days to travel its entire length!

The word *Mississippi* is also used for a slightly different purpose: to *count mississippily*.

If you're not familiar with the phrase, we're here to explain to you what it means: it's used to count seconds.

The idea behind it is that adding the word *Mississippi* to a number when counting seconds aloud makes them sound closer to clock-time, and therefore "one Mississippi, two Mississippi, three Mississippi" will take approximately an actual three seconds of time! It's often used by children playing hide-and-seek to make sure the seeker does an honest count.

Your task is very simple here: write a program that uses a `for` loop to "count mississippily" to five. Having counted to five, the program should print to the screen the final message `"Ready or not, here I come!"`

Use the skeleton we've provided in the editor.

EXTRA INFO

Note that the code in the editor contains two elements which may not be fully clear to you at this moment: the `import time` statement, and the `sleep()` method. We're going to talk about them soon.

For the time being, we'd just like you to know that we've imported the `time` module and used the `sleep()` method to suspend the execution of each subsequent `print()` function inside the `for` loop for one second, so that the message outputted to the console resembles an actual counting. Don't worry - you'll soon learn more about modules and methods.

Expected output

```
1 Mississippi
2 Mississippi
3 Mississippi
4 Mississippi
5 Mississippi
Ready or not, here I come!
```

```
import time


# Write a for loop that counts to five.
    # Body of the loop - print the loop iteration number and the word "Mississippi".
    # Body of the loop - use: time.sleep(1)


# Write a print function with the final message.
```

LAB 13: Break

Estimated time

10-20 minutes

Level of difficulty

Easy

Objectives

Familiarize the student with:

- using the `break` statement in loops;
- reflecting real-life situations in computer code.

Scenario

The `break` statement is used to exit/terminate a loop.

Design a program that uses a `while` loop and continuously asks the user to enter a word unless the user enters `"chupacabra"` as the secret exit word, in which case the message `"You've successfully left the loop."` should be printed to the screen, and the loop should terminate.

Don't print any of the words entered by the user. Use the concept of conditional execution and the `break` statement.

LAB 14: Continue

Estimated time

10-20 minutes

Level of difficulty

Easy

Objectives

Familiarize the student with:

- using the `continue` statement in loops;
- reflecting real-life situations in computer code.

Scenario

The `continue` statement is used to skip the current block and move ahead to the next iteration, without executing the statements inside the loop.

It can be used with both the `while` and `for` loops.

Your task here is very special: you must design a vowel eater! Write a program that uses:

- a `for` loop;
- the concept of conditional execution (*if-elif-else*)
- the `continue` statement.

Your program must:

- ask the user to enter a word;
- use `word = word.upper()` to convert the word entered by the user to upper case; we'll talk about the so-called **string methods** and the `upper()` method very soon - don't worry;
- use conditional execution and the `continue` statement to "eat" the following vowels *A*, *E*, *I*, *O*, *U* from the inputted word;
- print the uneaten letters to the screen, each one of them on a separate line.

Test your program with the data we've provided for you.

Test data

Sample input: `Gregory`

Expected output:

```
G
R
G
R
Y
```

Sample input: `abstemious`

Expected output:

```
B
S
T
M
S
```

LAB 15: Pyramid

Estimated time

20-30 minutes

Level of difficulty

Medium

Objectives

Familiarize the student with:

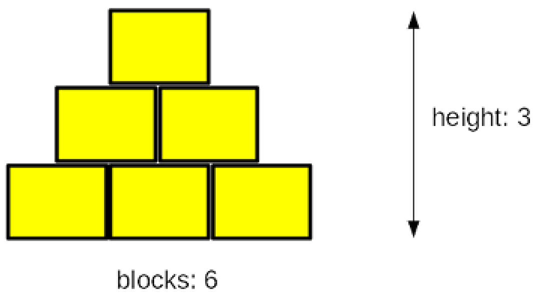
- using the `while` loop;
- finding the proper implementation of verbally defined rules;
- reflecting real-life situations in computer code.

Scenario

Listen to this story: a boy and his father, a computer programmer, are playing with wooden blocks. They are building a pyramid.

Their pyramid is a bit weird, as it is actually a pyramid-shaped wall - it's flat. The pyramid is stacked according to one simple principle: each lower layer contains one block more than the layer above.

The figure illustrates the rule used by the builders:



Your task is to write a program which reads the number of blocks the builders have, and outputs the height of the pyramid that can be built using these blocks.

Note: the height is measured by the number of **fully completed layers** - if the builders don't have a sufficient number of blocks and cannot complete the next layer, they finish their work immediately.

Test your code using the data we've provided.

Test Data

Sample input: `6`

Expected output: `The height of the pyramid: 3`

Sample input: `20`

Expected output: `The height of the pyramid: 5`

Sample input: `1000`

Expected output: `The height of the pyramid: 44`

Sample input: `2`

Expected output: `The height of the pyramid: 1`

LAB 16: School Admission

Estimated time

20 minutes

Level of difficulty

Easy

Objectives

Learn how to use logical operators (and, or, not) in Python by solving simple problems.

Scenario

Logical operators are used to combine conditional statements:

- **and**: Returns True if both statements are true.
- **or**: Returns True if at least one statement is true.
- **not**: Reverses the result, returns False if the result is true.

Tasks:

1. **Determine Eligibility for a Discount:**
   a. A store offers a discount if a customer is either a member or has spent more than $100.
   b. Customers also get an additional discount if they are both a member and have spent more than $100.
2. **Check Admission Criteria:**
   a. A student qualifies for admission if they have scored more than 70 in both Math and Science or if they have scored more than 90 in either subject.
   b. If they scored English and math or science more than 70 they are qualified

LAB 17: Basic List

Estimated time

5 minutes

Level of difficulty

Very easy

Objectives

Familiarize the student with:

- using basic instructions related to lists;
- creating and modifying lists.

Scenario

There once was a hat. The hat contained no rabbit, but a list of five numbers: `1`, `2`, `3`, `4`, and `5`.

Your task is to:

- write a line of code that prompts the user to replace the middle number in the list with an integer number entered by the user (Step 1)
- write a line of code that removes the last element from the list (Step 2)
- write a line of code that prints the length of the existing list (Step 3).

Ready for this challenge?

LAB 18: The Beatles

Estimated time

10-15 minutes

Level of difficulty

Easy

Objectives

Familiarize the student with:

- creating and modifying simple lists;
- using methods to modify lists.

Scenario

The Beatles were one of the most popular music group of the 1960s, and the best-selling band in history. Some people consider them to be the most influential act of the rock era. Indeed, they were included in *Time* magazine's compilation of the 20th Century's 100 most influential people.

The band underwent many line-up changes, culminating in 1962 with the line-up of John Lennon, Paul McCartney, George Harrison, and Richard Starkey (better known as Ringo Starr).

Write a program that reflects these changes and lets you practice with the concept of lists. Your task is to:

- step 1: create an empty list named `beatles`;
- step 2: use the `append()` method to add the following members of the band to the list: `John Lennon`, `Paul McCartney`, and `George Harrison`;
- step 3: use the `for` loop and the `append()` method to prompt the user to add the following members of the band to the list: `Stu Sutcliffe`, and `Pete Best`;
- step 4: use the `del` instruction to remove `Stu Sutcliffe` and `Pete Best` from the list;
- step 5: use the `insert()` method to add `Ringo Starr` to the beginning of the list.

By the way, are you a Beatles fan? (The Beatles is one of Greg's favorite bands. But wait...who's Greg...?)

LAB 19: Operator on a list

Estimated time

10-15 minutes

Level of difficulty

Easy

Objectives

Familiarize the student with:

- list indexing;
- utilizing the `in` and `not in` operators.

Scenario

Imagine a list - not very long, not very complicated, just a simple list containing some integer numbers. Some of these numbers may be repeated, and this is the clue. We don't want any repetitions. We want them to be removed.

Your task is to write a program which removes all the number repetitions from the list. The goal is to have a list in which all the numbers appear not more than once.

Note: assume that the source list is hard-coded inside the code - you don't have to enter it from the keyboard. Of course, you can improve the code and add a part that can carry out a conversation with the user and obtain all the data from her/him.

Hint: we encourage you to create a new list as a temporary work area - you don't need to update the list in situ.

We've provided no test data, as that would be too easy. You can use our skeleton instead.

```
my_list = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]

#

# Write your code here.

#

print("The list with unique elements only:")

print(my_list)
```

LAB 20: Leap Year Function

Estimated time

10-15 minutes

Level of difficulty

Easy

Objectives

Familiarize the student with:

- projecting and writing parameterized functions;
- utilizing the `return` statement;
- testing the functions.

Scenario

Your task is to write and test a function which takes one argument (a year) and returns `True` if the year is a *leap year*, or `False` otherwise.

The seed of the function is already shown in the skeleton code in the editor.

Note: we've also prepared a short testing code, which you can use to test your function.

The code uses two lists – one with the test data, and the other containing the expected results. The code will tell you if any of your results are invalid.

```
# Testing the function
test_data = [1900, 2000, 2016, 1987]
test_results = [False, True, True, False]

for i in range(len(test_data)):
    yr = test_data[i]
    print(yr, "->", end="")
    result = is_leap_year(yr)
    if result == test_results[i]:
        print("OK")
    else:
        print("Failed")
```

LAB 20.1: Leap Year Function

Estimated time

15-20 minutes

Level of difficulty

Medium

Prerequisites

LAB 4.3.1.6

Objectives

Familiarize the student with:

- projecting and writing parameterized functions;
- utilizing the `return` statement;
- utilizing the student's own functions.

Scenario

Your task is to write and test a function which takes two arguments (a year and a month) and returns the number of days for the given month/year pair (while only February is sensitive to the `year` value, your function should be universal).

The initial part of the function is ready. Now, convince the function to return `None` if its arguments don't make sense.

Of course, you can (and should) use the previously written and tested function (LAB 4.3.1.6). It may be very helpful. We encourage you to use a list filled with the months' lengths. You can create it inside the function - this trick will significantly shorten the code.

We've prepared a testing code. Expand it to include more test cases.

```
# Testing the function
test_years = [1900, 2000, 2016, 1987, 2020, 2021, 1800, 2400]
test_months = [2, 2, 1, 11, 2, 2, 2, 2]
test_results = [28, 29, 31, 30, 29, 28, 28, 29]

for i in range(len(test_years)):
    yr = test_years[i]
    mo = test_months[i]
    print(yr, mo, "->", end="")
    result = days_in_month(yr, mo)
    if result == test_results[i]:
        print("OK")
    else:
        print("Failed")
```

LAB 20.2: Leap Year Function

Estimated time

20-30 minutes

Level of difficulty

Medium

Prerequisites

LAB 4.3.1.6
LAB 4.3.1.7

Objectives

Familiarize the student with:

- projecting and writing parameterized functions;
- utilizing the `return` statement;
- building a set of utility functions;
- utilizing the student's own functions.

Scenario

Your task is to write and test a function which takes three arguments (a year, a month, and a day of the month) and returns the corresponding day of the year, or returns `None` if any of the arguments is invalid.

Use the previously written and tested functions. Add some test cases to the code. This test is only a beginning.

LAB 21: Prime Number

Estimated time

15-20 minutes

Level of difficulty

Medium

Objectives

- familiarizing the student with classic notions and algorithms;
- improving the student's skills in defining and using functions.

Scenario

*A natural number is **prime** if it is greater than 1 and has no divisors other than 1 and itself.*

Complicated? Not at all. For example, 8 isn't a prime number, as you can divide it by 2 and 4 (we can't use divisors equal to 1 and 8, as the definition prohibits this).

On the other hand, 7 is a prime number, as we can't find any legal divisors for it.

Your task is to write a function checking whether a number is prime or not.

The function:

- is called `is_prime`;
- takes one argument (the value to check)
- returns `True` if the argument is a prime number, and `False` otherwise.

Hint: try to divide the argument by all subsequent values (starting from 2) and check the remainder - if it's zero, your number cannot be a prime; think carefully about when you should stop the process.

If you need to know the square root of any value, you can utilize the `**` operator. Remember: the square root of x is the same as $x^{0.5}$

Complete the code in the editor.

Run your code and check whether your output is the same as ours.

Expected output

```
2 3 5 7 11 13 17 19
```

LAB 22: Tuple

Estimated time

20 minutes

Level of difficulty

Easy

Objectives

To understand the basic operations performed on tuples in Python.

Scenario

Tasks:

Tuple Creation:

- Create a tuple named fruits containing the following fruits: "apple", "banana", "orange", "kiwi", and "grape".

Accessing Elements:

- Print the third element of the tuple fruits.
- Print the last element of the tuple fruits.

Tuple Length:

- Find and print the length of the tuple fruits.

Slicing:

- Create a new tuple named selected_fruits containing the first three fruits from the fruits tuple.
- Print the selected_fruits tuple.

Adding Element:

- Create another tuple named more_fruits containing the following fruits: "pineapple" and "watermelon".
- Concatenate the fruits and more_fruits tuples and store the result in a new tuple named all_fruits.
- Print the all_fruits tuple.

Repetition:

- Repeat the fruits tuple three times and store the result in a new tuple named repeated_fruits.
- Print the repeated_fruits tuple.

Membership Test:

- Check if "orange" is present in the fruits tuple and print the result.

Immutable Nature:

- Attempt to change the third element of the fruits tuple to "pear".
- Print the fruits tuple to see if the modification was successful.

Bonus Task (Unpacking):

- Unpack the fruits tuple into separate variables named fruit1, fruit2, fruit3, fruit4, and fruit5.
- Print each variable.

LAB 23: Dictionary

Estimated time

20 minutes

Level of difficulty

Easy

Objectives

To understand the basic operations performed on dictionaries in Python.

Scenario

Tasks:

Dictionary Creation:

- Create a dictionary named student with the following key-value pairs:
- "name": "John"
- "age": 20
- "gender": "Male"
- "major": "Computer Science"
- "GPA": 3.8

Accessing Elements:

- Print the value associated with the key "age" from the student dictionary.
- Print the value associated with the key "GPA" from the student dictionary.

Dictionary Length:

- Find and print the number of key-value pairs in the student dictionary.

Adding a New Key-Value Pair:

- Add a new key-value pair to the student dictionary:
- Key: "year"
- Value: 3

Updating a Value:

- Update the value of the key "GPA" in the student dictionary to 3.9.

Removing a Key-Value Pair:

- Remove the key-value pair with the key "major" from the student dictionary.

Key Existence Check:

- Check if the key "major" exists in the student dictionary and print the result.

Printing Keys and Values:

- Print all keys in the student dictionary.
- Print all values in the student dictionary.

Bonus Task (Clearing the Dictionary):

- Clear all key-value pairs from the student dictionary.
- Print the student dictionary to confirm it is empty.

LAB 24: Set

Estimated time

20 minutes

Level of difficulty

Easy

Objectives

To understand the basic operations performed on sets in Python.

Scenario

Tasks:

Set Creation:

- Create a set named A containing the following elements: 1, 2, 3, 4, 5.

Set Length:

- Find and print the number of elements in set A.

Adding Elements:

- Add the element 6 to set A.
- Add the element 7 to set A.

Removing Elements:

- Remove the element 3 from set A.

Set Operations:

- Create another set named B containing the elements: 4, 5, 6, 7, 8.
- Find and print the union of sets A and B.
- Find and print the intersection of sets A and B.
- Find and print the difference between sets A and B.
- Find and print the symmetric difference between sets A and B.

Subset Check:

- Check if set A is a subset of set B and print the result.

Bonus Task (Clearing the Set):

- Clear all elements from set A.
- Print set A to confirm it is empty.

LAB 25: Split Function

Estimated time

20-25 minutes

Level of difficulty

Medium

Objectives

- improving the student's skills in operating with strings;
- using built-in Python string methods.

Scenario

You already know how `split()` works. Now we want you to prove it.

Your task is to **write your own function, which behaves almost exactly like the original** `split()` **method**, i.e.:

- it should accept exactly one argument - a string;
- it should return a list of words created from the string, divided in the places where the string contains whitespaces;
- if the string is empty, the function should return an empty list;
- its name should be `mysplit()`

Use the template in the editor. Test your code carefully.

Expected output

```
['To', 'be', 'or', 'not', 'to', 'be,', 'that', 'is', 'the', 'question']

['To', 'be', 'or', 'not', 'to', 'be,that', 'is', 'the', 'question']

[]

['abc']

[]
```

LAB 26: Exception

Estimated time

15-25 minutes

Level of difficulty

Medium

Objectives

- improving the student's skills in defining functions;
- using exceptions in order to provide a safe input environment.

Scenario

Your task is to write a **function able to input integer values and to check if they are within a specified range**.

The function should:

- accept three arguments: a prompt, a low acceptable limit, and a high acceptable limit;
- if the user enters a string that is not an integer value, the function should emit the message `Error: wrong input`, and ask the user to input the value again;
- if the user enters a number which falls outside the specified range, the function should emit the message `Error: the value is not within permitted range (min..max)` and ask the user to input the value again;
- if the input value is valid, return it as a result.

Test data

Test your code carefully.

This is how the function should react to the user's input:

```
Enter a number from -10 to 10: 100

Error: the value is not within permitted range (-10..10)

Enter a number from -10 to 10: asd

Error: wrong input

Enter number from -10 to 10: 1

The number is: 1
```

LAB 27: Directory

Estimated time

15-30 min

Level of difficulty

Easy

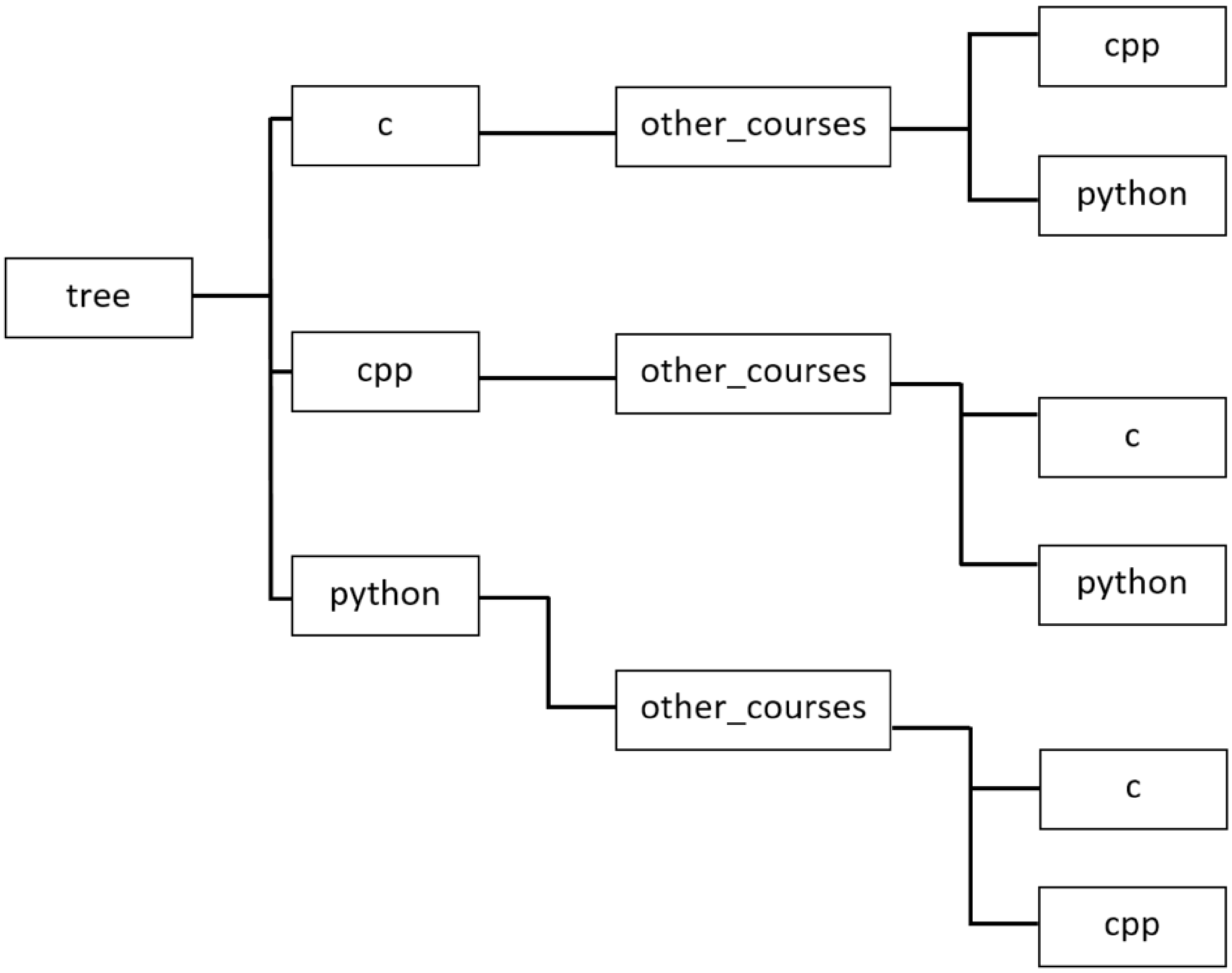
Objectives

- improving the student's skills in interacting with the operating system;
- practical use of known functions provided by the *os* module.

Scenario

It goes without saying that operating systems allow you to search for files and directories. While studying this part of the course, you learned about the functions of the *os* module, which have everything you need to write a program that will search for directories in a given location.

To make your task easier, we have prepared a test directory structure for you:



Your program should meet the following requirements:

1. Write a function or method called *find* that takes two arguments called *path* and *dir*. The *path* argument should accept a relative or absolute path to a directory where the search should start, while the *dir* argument should be the name of a directory that you want to find in the given path. Your program should display the absolute paths if it finds a directory with the given name.
2. The directory search should be done recursively. This means that the search should also include all subdirectories in the given path.

Example input:

```
path="./tree", dir="python"
```

Example output:

```
.../tree/python
```

```
.../tree/cpp/other_courses/python
```

```
.../tree/c/other_courses/python
```