Python Programming 1

(CP1S02005E)

Białystok University of Technology
Faculty of Electrical Engineering

Industry Digitization, semester II
Academic year 2024/2025

Lecture no. 01 (05.03.2025)

Jarosław Forenc, PhD

Basic information

- Jarosław Forenc, PhD
- Bialystok University of Technology, Faculty of Electrical Engineering Department of Electrotechnics, Power Electronics and Electrical Power Engineering
- Wiejska 45D Street, 15-351 Bialystok
  room: WE-204
- e-mail: j.forenc@pb.edu.pl

- http://jforenc.prv.pl/pprog1.html
  - course materials
- Office hours (consultations):
  - Monday, 08:30-10:00, room WE-204
  - Wednedsay, 08:30-10:00, room WE-204

Module content

1. General structure of a Python program. Data types, keywords, and variable names. Arithmetic operators and expressions, operator precedence, mathematical functions. Comments. Comparison and logical operators, logical expressions. Conditional statements if/elif/else.
2. Iterative statements: for and while, the range function, break and continue statements.
3. Strings (text type), string operations (methods). Lists, tuples, dictionaries, and sets.
4. Functions, defining functions, arguments and parameters, variable scope.
5. File operations, exceptions.
6. Object-oriented programming, objects, classes, inheritance.
7. Standard library, NumPy, Matplotlib, SciPy libraries. Jupyter Notebook environment.
8. Final exam.

Literature

1. Ramalho L., Fluent Python: clear, concise, and effective programming. Sebastopol, O'Reilly, 2022.
2. Matthes E., Python Crash Course, San Francisco, CA, No Starch Press, 2019.
3. Sweigart A., Automate the Boring Stuff with Python, San Francisco, CA, No Starch Press, 2020.
4. Lutz M., Learning Python, Sebastopol, CA, O'Reilly Media, 2013.
5. https://www.python.org/doc/ - Python, documentation.

Learning outcomes

The basis for passing the course (earning ECTS points) is confirming that each of the assumed learning outcomes has been achieved.

A student who has passed the course knows and understands:

EU1	the fundamental mechanisms of the Python language that enable structural and object-oriented programming
EU2	basic programming constructs used in Python

- Details: http://jforenc.prv.pl/pprog1.html or USOS system

Lecture assessment

- The lecture assessment will be based on the results of a written test
- The test will take place during the last lecture of the semester
- The test score can range from 0 to 100 points
- The final grade is based on the points earned:

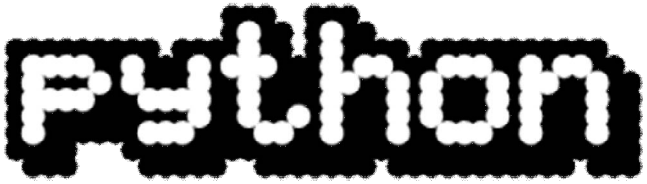
Points	Grade	Points	Grade
91 - 100	5,0	61 - 70	3,5
81 - 90	4,5	51 - 60	3,0
71 - 80	4,0	0 - 50	2,0

Topics

- Basic information
- First program
- Variables, keywords, variable types
- print() and input() functions
- Arithmetic operators, operator precedence
- Constants and mathematical functions
- Comments
- Comparison and logical operators
- if, if-else, if-elif-else statements
- Conditional operator

Python - Information

- Website: http://python.org
- Current stable version: 3.13.2 (04.02.2025)
- Operating systems: Windows, macOS, Linux, Android, Unix, BSD and others
- Implementations: CPython, PyPy, Stackless Python, MicroPython, CircuitPython, IronPython, Jython
- Logo:



1997 - 2006 2006 - now

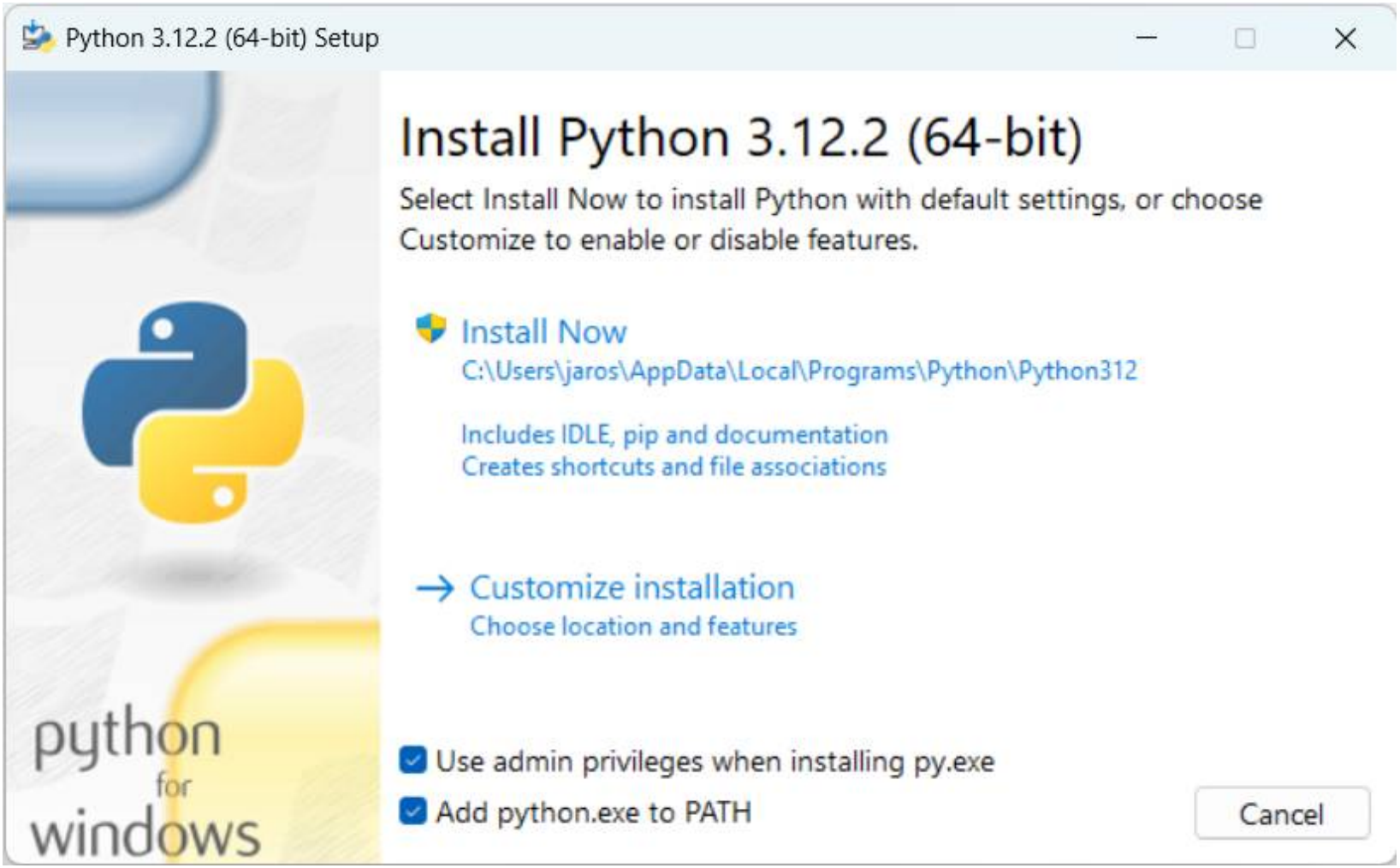
Python - First program

- A simple text file with a .py extension
- Example code:

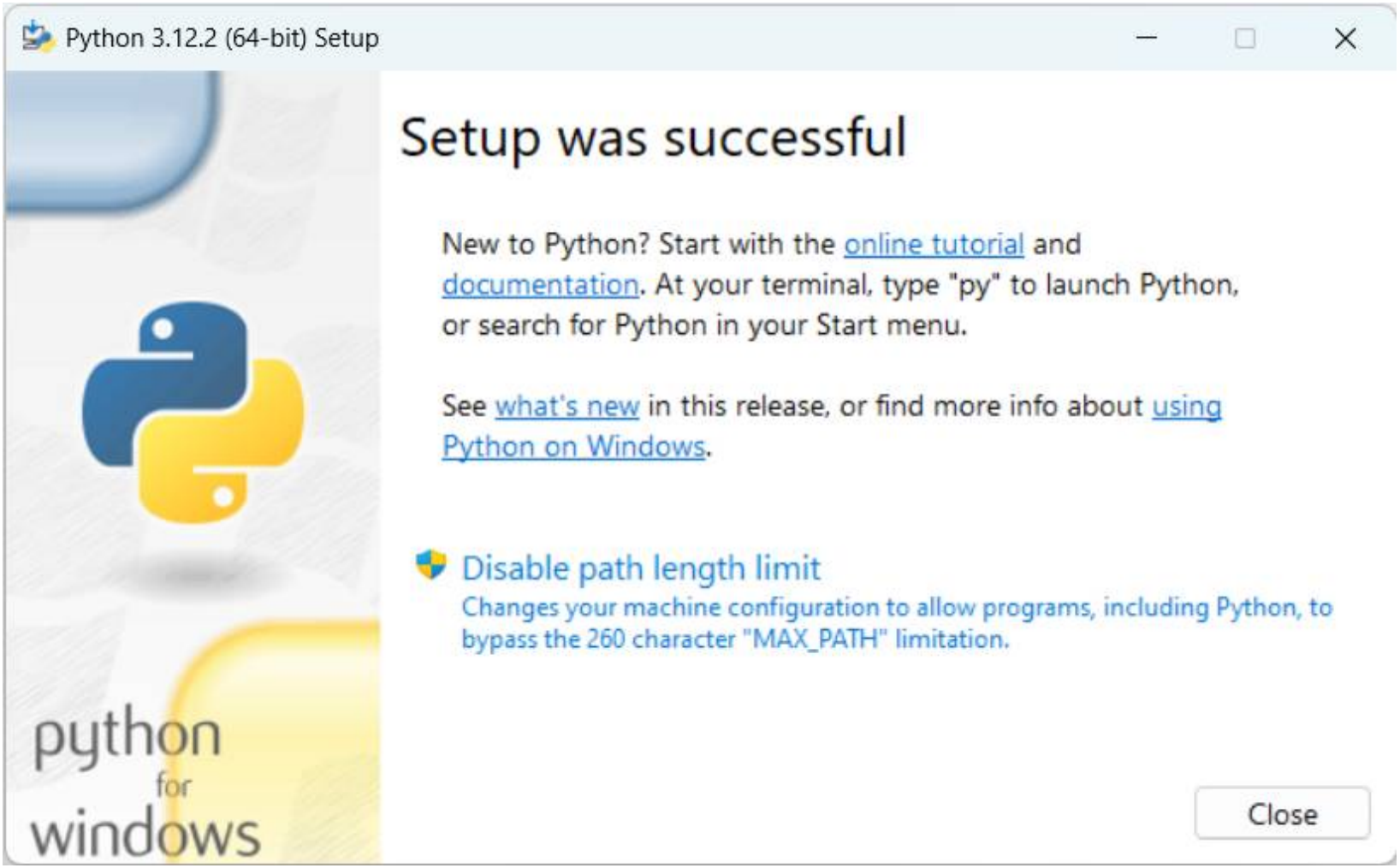
```
print("Hello, World!")
```

- The print() function is used to display text or other data in the console
- Running the program requires Python to be installed
  - https://www.python.org/downloads/windows/
  - Python 3.13.2: https://www.python.org/downloads/release/python-3132/
  - Windows installer (64-bit) - file 27.3 MB

Python - Installation



Python - Installation

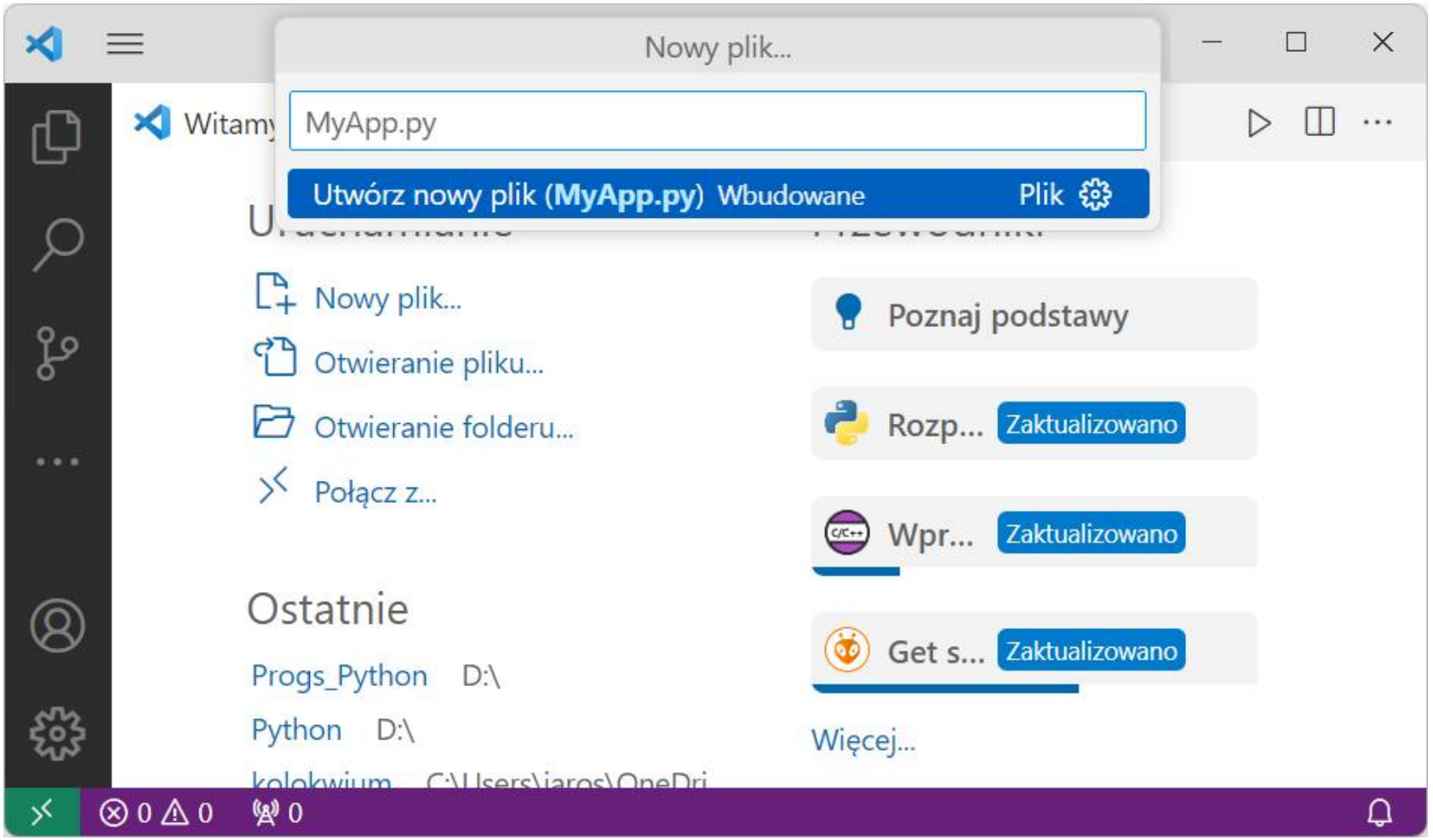


Microsoft Visual Studio Code

- Installation of the Python extension (Microsoft)

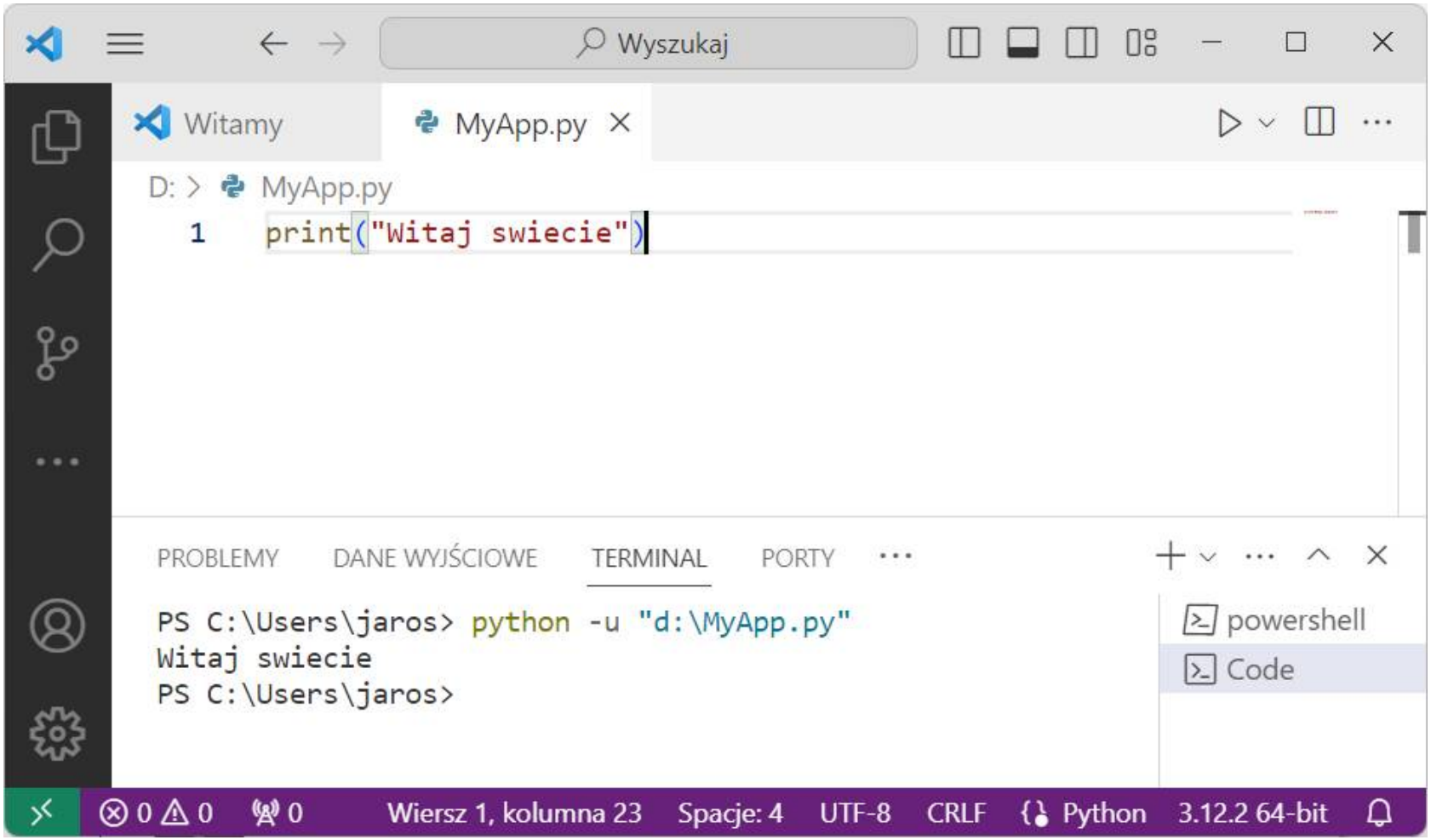
Microsoft Visual Studio Code

- We add a new file (with the .py extension)



Microsoft Visual Studio Code

- We enter the code and run the program



Python - Variables

- Declaring variables involves assigning a value to a specific identifier
- The variable type is determined dynamically based on the assigned value (there is no need for explicit type declaration)

```
a = 10
b = "Hello"
c = 3.14
d = 5.2e-6
is_true = True
```

- Multiple value assignments to variables in a single line

```
a, b, c = 10, 20, 30
```

Python - Variable names

- Rules and Guidelines:
  - variable names can only contain letters, digits and the underscore (_)
  - a variable name can start with a letter or an underscore but not with a digit

```
temp2    _variable4
```

```
2temp    my-variable
```

  - spaces are not allowed in variable names (however, underscores can be used to separate words)

```
my_variable    circle_area    temp_celsius
```

  - avoid using Python keywords as variable names

```
print    class    input    pass
```

Python - Keywords

- Keywords are written in lower case

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Python - Variable names

- Rules and Guidelines :
  - the variable name should be short but readable

```
tc      tempc
```

```
temperature_in_celsius
```

  - use lowercase letter l and uppercase letter O with caution, as they can be easily mistaken for the digits 1 and 0
  - variable names should be written in lowercase, as it is conventionally accepted that uppercase letters are used for constants
- There is no special way to write or define constants

```
MAX = 9999
```

```
(constant) MAX: Literal[9999]
MAX = 9999
```

```
(variable) max: Literal[9999]
max = 9999
```

Python - print() function

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
      flush=False)
```

- objects - objects to be displayed; one or more arguments (text, numbers, lists, variables, etc.)
- sep (optional) - separator between objects; by default, it is a single space
- end (optional) - line-ending character; by default, it is '\n' - each call to print() ends with a newline; assigning end='' allows you to call print() without moving the cursor to a new line
- file (optional) - the file object where the text should be written; by default, this is standard output (sys.stdout)
- flush (optional) - a boolean value indicating whether to force the buffer to be flushed; by default, it is False

Python - print() function

```
print("Hello world!")
```

```
Hello world!
```

```
print("Hello", "world", sep=", ", end="!\n")
```

```
Hello, world!
```

```
x = 10
y = 20
print("Value x is", x, "value y is", y)
```

```
Value x is 10 value y is 20
```

Python - print() functions, f-strings

```
name = "Kate"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Kate and I am 40 years old.
```

```
x = 1.23456789
print(f"Value x = {x:.3f}")
```

```
Value x = 1.235
```

```
x = 1.2345
print(f"Value x = [{x:10.2f}]")
```

```
Value x = [      1.23]
```

Python - print() functions, f-strings

```
x = 1.2345
print(f"Value x = [{x:>10.2f}]")
```

```
Value x = [      1.23]
```

```
x = 1.2345
print(f"Value x = [{x:<10.2f}]")
```

```
Value x = [1.23      ]
```

```
x = 1.2345
print(f"Value x = [{x:^10.2f}]")
```

```
Value x = [   1.23   ]
```

Python - Data types

- Integers:
  - used number systems: decimal, binary, octal, hexadecimal

```
w, x, y, z = 10, 0b10, 0o10, 0x10
print(w,x,y,z)
```

```
10 2 8 16
```

  - displaying numbers in different number systems

```
x = 255
print(f"Binary: {x:b}")
print(f"Octal: {x:o}")
print(f"Hexadecimal: {x:x}")
print(f"Hexadecimal: {x:X}")
```

```
Binary: 11111111
Octal: 377
Hexadecimal: ff
Hexadecimal: FF
```

Python - Data types

- Integers:
  - when dividing integers, the result is always a floating-point number

```
x = 4/2
print(x)
```

```
2.0
```

  - when writing numbers consisting of multiple digits, you can group them using underscores

```
population = 8_019_000_000
print(population)
```

```
8019000000
```

Python - Data types

- Floating-point numbers:
  - in the case of operations involving an integer and a real number, the result is always a floating-point number

```
x = 2 + 2.0
print(x)
```

```
4.0
```

  - sometimes, the results of operations on floating-point numbers can be surprising

```
x = 0.2 + 0.1
print(x)
```

```
0.30000000000000004
```

Python - Data types

- String sequences :
  - a string can be enclosed in either double quotes or single quotes

```
txt1 = "Kate has a laptop"
txt2 = 'John has a tablet'
```

- The built-in function type() allows you to check the type of the given argument

```
print(type(32))
print(type(2.5))
print(type(3.1e-4))
print(type(True))
print(type("text"))
print(type('text'))
```

```
<class 'int'>
<class 'float'>
<class 'float'>
<class 'bool'>
<class 'str'>
<class 'str'>
```

Python - input() function

```
input(prompt)
```

- the input() function is used to get data from the user via the keyboard
- prompt (optional) - text displayed before waiting for the user to enter data; it provides information on what the user should type
- the input() function returns the data entered by the user as a string (type str)
- when entering numbers, it is necessary to use a conversion function, such as int() or float()

Python - input() function

```
name = input("What is your name: ")
print("Hello,", name)
```

```
What is your name: Paul
Hello, Paul
```

```
year = input("Enter your year of birth: ")
age = 2024 - year
print("You are", age, "years old")
```

```
Enter your year of birth: 2003
Traceback (most recent call last):
  File "d:\myapp.py", line 2, in <module>
    age = 2024 - year
          ~~~~~^~~~~
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Python - input() function

```
year = int(input("Enter your year of birth: "))
age = 2024 - year
print("You are", age, "years old")
```

```
Enter your year of birth: 1997
You are 27 years old
```

```
Tf = float(input("Enter temperature [F]: "))
print(f"{Tf} [F] is {5/9*(Tf-32)} [C]")
```

```
Enter temperature [F]: 45.5
45.5 [F] is 7.5 [C]
```

Python - Arithmetic operators

- Addition (+): adds two numbers or joins two strings

```
x = 5
y = 3
print(x + y)  # Will display: 8
```

- Subtraction (-): subtracts one number from another

```
x = 5
y = 3
print(x - y)  # Will display: 2
```

Python - Arithmetic operators

- Multiplication (*): multiplies two numbers

```
x = 5
y = 3
print(x * y)  # Will display: 15
```

- Division (/): divides one number by another, returns a floating point number

```
x = 5
y = 3
print(x / y)  # Will display: 1.6666666666666667
```

Python - Arithmetic operators

- Integer division (//): Divides one number by another and returns the result as an integer, rounding down

```
x = 5
y = 3
print(x // y)  # Will display: 1
```

- Remainder of Division (%): returns the remainder of dividing one number by another

```
x = 5
y = 3
print(x % y)  # Will display: 2
```

Python - Arithmetic operators

- Power (**): Raises the first number to the power of the second

```
x = 5
y = 3
print(x ** y)  # Will display: 125
```

Python - Operator precedence

- Operator precedence from highest to lowest:
  - expressions in parentheses - expressions within parentheses are always evaluated first
  - power (**) - the power operator has the highest precedence
  - unary operators: negative (-) and positive (+) - these operators are evaluated before other arithmetic operators
  - multiplication (*), division (/), integer division (//), and remainder of division (%) - these operators are evaluated before addition and subtraction
  - addition (+) and subtraction (-) - the addition and subtraction operators have the lowest precedence
- Arithmetic operators are left-associative

```
x = 5 - 2 + 3      # (-) → (+) → (=)
```

Python - Augmented assignment operators

Operator	Example of statement	Equivalent statement
`+=`	`x += 10`	`x = x + 10`
`-=`	`x -= 10`	`x = x - 10`
`*=`	`x *= 10`	`x = x * 10`
`/=`	`x /= 10`	`x = x / 10`
`//=`	`x //= 10`	`x = x // 10`
`%=`	`x %= 10`	`x = x % 10`
`**=`	`x **= 10`	`x = x ** 10`

Python - Mathematical constants and functions

- Mathematical constants and functions are available in the math module
- To use these functions and constants, you must first import the module

```
import math
```

- Mathematical constants:
  - math.pi - the constant π
  - math.e - the constant e (the base of the natural logarithm)
  - math.inf - infinity (positive)
  - math.nan - NaN (Not a Number) - represents an undefined or unrepresentable numerical value

Python - Mathematical constants and functions

- Mathematical functions:
  - sin(x) - calculates the sine of angle x (in radians)
  - cos(x) - calculates the cosine of angle x (in radians)
  - tan(x) - calculates the tangent of angle x (in radians)
  - atan(x) - calculates the arctangent of x in radians, returning a result in the range $[-\pi/2, \pi/2]$
  - sqrt(x) - calculates the square root of x
  - exp(x) - calculates the exponential value of e raised to the power of x
  - log(x) - calculates the natural logarithm of x
  - log10(x) - calculates the base-10 logarithm of x
  - pow(x, y) - calculates x raised to the power of y
  - fabs(x) - returns the absolute value |x|

Python - Example

```
import math

# Reading the radius of a circle from the keyboard
radius = float(input("Enter the radius: "))

# Calculation of the area of a circle
area = math.pi * math.pow(radius,2)

# Calculation of the circumference of a circle
circumf = 2 * math.pi * radius

# Displaying the results
print(f"Circle area: {area:.2f}")
print(f"Circumf: {circumf:.2f}")
```

```
Enter the radius: 10
Circle area: 314.16
Circumf: 62.83
```

Python - Comments

- Comments are used to add descriptions or explanations in the source code of a program and are ignored during program execution
- They start with the # symbol and cover the entire line
- Everything after the # symbol is treated as a comment

```
# The code of the simplest program
print("Hello world!")
```

Python - Comparison operators

Operator	Meaning	Example	Result
==	equal to	`2 == 2`	`True`
!=	not equal to	`2 != 2`	`False`
>	greater than	`2 > 5`	`False`
<	less than	`2 < 5`	`True`
>=	greater than or equal to	`2 >= 5`	`False`
<=	less than or equal to	`2 <= 2`	`True`

- as a result of a comparison, we get a value of True or False
- the values True and False can be assigned to variables

```
result = True
condition = False
```

Python - Comparison operators (examples)

```
age = 17
print(age == 18)
```

```
False
```

```
age = 17
print(age < 18)
```

```
True
```

```
age = 17
print(age >= 15)
```

```
True
```

Python - Comparison operators (examples)

```
name = "John"
print(name != "John")
```

```
False
```

```
name = "John"
print(name == "John")
```

```
True
```

```
name = "John"
print(name > "Elizabeth")
```

```
True
```

Python - Comparison operators (examples)

```
name = "John"
print(name == "john")
```

```
False
```

```
name = "John"
print(name.lower() == "john")
```

```
True
```

- the lower() method returns a string in which all uppercase letters have been converted to lowercase
- this method does not change the value of the variable name

Python - Logical operators

Operator	Meaning	Example
`and`	returns True if both conditions are True	`x > 5 and x <= 7`
`or`	returns True if at least one of the conditions is True	`x < 3 or x >= 4`
`not`	returns the inverse of the logical value of the expression	`not x`

```
age1 = 19
age2 = 17
print(age1 >= 18 and age2 >= 18)
```

```
False
```

Python - Logical operators (examples)

- you can use additional brackets in the condition, but it is not necessary

```
age1 = 19
age2 = 17
print((age1 >= 18) and (age2 >= 18))
```

- example of using the or operator

```
age1 = 19
age2 = 17
print(age1 >= 18 or age2 >= 18)
```

```
True
```

Python - if statement

- The simplest form of the if statement:

```
if conditional_test:
    any_action
```

  - if the result of conditional_test is True, then the action is performed; if it is False, then the action is not performed
  - the indented code block specifies the actions performed after the if

```
age = 19
if age >= 18:
    print("You are of legal age")
```

  - a code block consists of one or more consecutive lines of code with the same level of indentation
  - indentation is typically: 4 spaces (recommended), 2 spaces, tab

Python - if statement

- where the indentation ends, the conditional block also ends:

```
age = 19
if age >= 18:
    print("You are an adult")
    print("You can go to the elections")
print("End")
```

Python - if-else statement

- if-else statement syntax:

```
if conditional_test:
    action1
else:
    action2
```

  - if the result of conditional_test is True then only action1 is executed; if it is False then only action2 is executed

```
number = int(input("Enter number: "))
if number % 2 == 0:
    print(f"{number} - even number")
else:
    print(f"{number} - odd number")
```

Python - example (square root)

```
import math

x = float(input("Enter number: "))
if x >= 0:
    y = math.sqrt(x)
    print("Square root:", y)
else:
    print("Error! Negative number")
```

```
Enter number: 5
Square root: 2.23606797749979

Enter number: -3
Error! Negative number
```

Python - example (square root)

```
import math

x = float(input("Enter number: "))
if x >= 0:
    y = math.sqrt(x)
    print("Square root:", y)
else:
    print("Error! Negative number")
```

```
Enter number: a
Traceback (most recent call last):
  File "d:\MyApp.py", line 3, in <module>
    x = float(input("Enter number: "))
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: could not convert string to float: 'a'
```

Python - example (square root)

```
import math

try:
    x = float(input("Enter number: "))
    if x >= 0:
        y = math.sqrt(x)
        print("Square root:", y)
    else:
        print("Error! Negative number")
except ValueError:
    print("Error! Please enter a valid number")
```

```
Enter number: a
Error! Please enter a valid number
```

Python - instrukcja if-elif-else

- if-elif-else statement syntax:

```
if test1:
    acrion1
elif test2:
    action2
else:
    action3
```

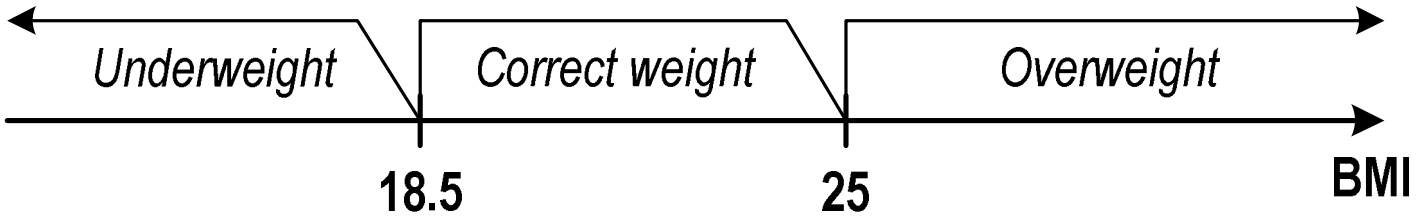
  - if the result of test1 is True then action1 is performed
  - if the result of test1 is False then test2 is checked
  - if the result of test2 is True then action2 is performed
  - if the result of test2 is False then action3 is performed
  - only one action can be performed

Python - example (BMI)

- BMI - coefficient obtained by dividing body weight in kilograms by the square of height in meters

$$BMI = \frac{weight}{height^2}$$

- For adults:
  - BMI < 18.5 - indicates underweight
  - BMI ≥ 18.5 and BMI < 25 - indicates a correct body weight
  - BMI ≥ 25 - indicates overweight

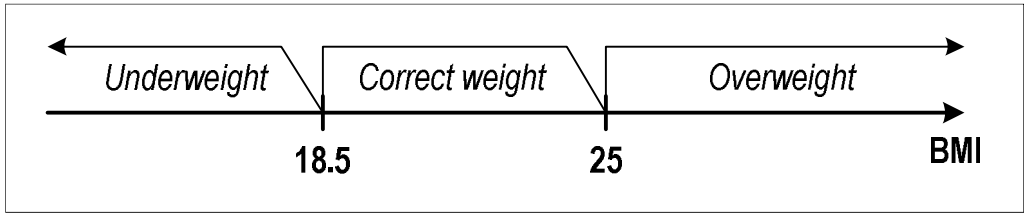


Python - example (BMI)

```
weight = float(input("Enter weight [kg]: "))
height = float(input("Enter height [m]: "))
bmi = weight / (height * height)
print("BMI:", "{:.2f}".format(bmi))

if bmi < 18.5:
    print("Underweight")
elif bmi >= 18.5 and bmi < 25:
    print("Correct weight")
else:
    print("Overweight")
```

```
Enter weight [kg]: 84
Enter height [m]: 1.85
BMI: 24.54
Correct weight
```



Python - example (BMI)

```
weight = float(input("Enter weight [kg]: "))
height = float(input("Enter height [m]: "))
bmi = weight / (height * height)
print("BMI:", "{:.2f}".format(bmi))

if bmi < 18.5:
    print("Underweight")
elif 18.5 <= bmi < 25:
    print("Correct weight")
else:
    print("Overweight")
```

- the condition checking whether bmi is within the range can also be written in another way

```
bmi >= 18.5 and bmi < 25
```

→

```
18.5 <= bmi < 25
```

Python - example (BMI)

```
weight = float(input("Enter weight [kg]: "))
height = float(input("Enter height [m]: "))
bmi = weight / (height * height)
print("BMI:", "{:.2f}".format(bmi))

if bmi < 18.5:
    print("Underweight")
else:
    if bmi < 25:
        print("Correct weight")
    else:
        print("Overweight")
```

- if statements can be nested

Python - example (grades)

```
pts = int(input("Enter number of points: "))

if pts < 51:
    grade = 2.0
elif pts < 61:
    grade = 3.0
elif pts < 71:
    grade = 3.5
elif pts < 81:
    grade = 4.0
elif pts < 91:
    grade = 4.5
else:
    grade = 5.0

print(f"Your grade: {grade:.1f}")
```

- the elif block can appear multiple times
- the final else block can be omitted

Python - Conditional operator

- Syntax of the conditional operator (ternary expression):

```
true_value if condition else false_value
```

  - condition is the logical condition to be checked
  - true_value is the value returned if the condition is met
  - false_value is the value returned if the condition is not met

```
x = int(input("Enter number: "))
txt = "Even" if x % 2 == 0 else "Odd"
print(txt)
```

End of lecture no. 1

Thank you for your attention!