

Introduction to programming

Labor01

Contact & Office Hours

► Piroska Biró, PhD

- biro.piroska@inf.unideb.hu
- **Office hours, IK-227:**
 - Tuesday 11–12
 - Thursday 14–15
- Group 6, Tuesday 8:00–10:00, TEOKJ II/108
- Group 9, Tuesday 12:00–14:00, TEOKJ II/112
- Group 4, Tuesday 14:00–16:00, TEOKJ II/112
- Group 5, Tuesday 16:00–18:00, TEOKJ II/112
- Group 7, Wednesday 8:00–10:00, IK-207
- Group 1, Wednesday 10:00–12:00, IK-207
- Group 8, Thursday 16:00–18:00, IK-207

Contact & Office Hours


► Marcell Beregi–Kovács

- beregi.kovacs.marcell@inf.unideb.hu
- **Office hours, IK–120:**
 - Tuesday 14:00–15:00
 - Wednesday 13:00–14:00
- Group 2, Tuesday 18:00–20:00, IK–106
- Group 3, Monday 18:00–20:00, TEOJK II/106

Topics

- ▶ The **goal of the subject** is to demonstrate what is programming and how computer programs are made.
- ▶ Presenting the **basic concepts** and constructs of programming it aims at building basic level programming skills.
- ▶ Building and developing **problem-solving and algorithmic skills** necessary for programming is also a key objective of the subject.
- ▶ The following main topics are covered using a high-level programming language that supports procedural programming (e.g., Python):

Topics

- ▶ Types, literals
 - ▶ Operators, expressions
 - ▶ Variables, assignment
 - ▶ Statements
 - ▶ Control structures
 - ▶ Basic data structures of the programming language of the course
 - ▶ Functions
 - ▶ Basic I/O
 - ▶ Other basic features of the programming language of the course
 - ▶ Writing simple programs
 - ▶ Steps of creating executable programs
 - ▶ Errors and bugs, debugging
 - ▶ Basic level use of developer tools (IDE)
- 

Requirements

- ▶ **Subject code:** INBPA0104–21
- ▶ **Semester:** 1
- ▶ **Type of course:** Labor
- ▶ **Number of Classes:** 0+0+2
- ▶ **Credit:** 3

- ▶ **Attendance and Participation:**
 - In every labor there will be an attendance sheet.
 - Maximum three absences are allowed in labor.
 - Maximum 15 minutes late arrival is accepted in labor.

- ▶ **Assessment:** Practical mark
 - Two test at the time of the labor (Midterm, Endterm)
- ▶ **Retakes:** in the first week of the examination period.
- ▶ **Materials:**
 - <https://elearning.unideb.hu/course/view.php?id=5313>

References

- ▶ Guido van Rossum: **Python Tutorial**
<https://docs.python.org/3/download.html>
- ▶ Wesley J. Chun: **Core Python Programming** (2nd Edition), 2006
- ▶ Allen B. Downey: **Think Python (How to Think Like a Computer Scientist)**
<http://www.greenteapress.com/thinkpython/>, O'Reilly, 2012
- ▶ Doug Hellmann: **The Python Standard Library by Example** (Developer's Library), 2011 (online: Python Module of the Week
<https://pymotw.com/3/>)

Python programming language

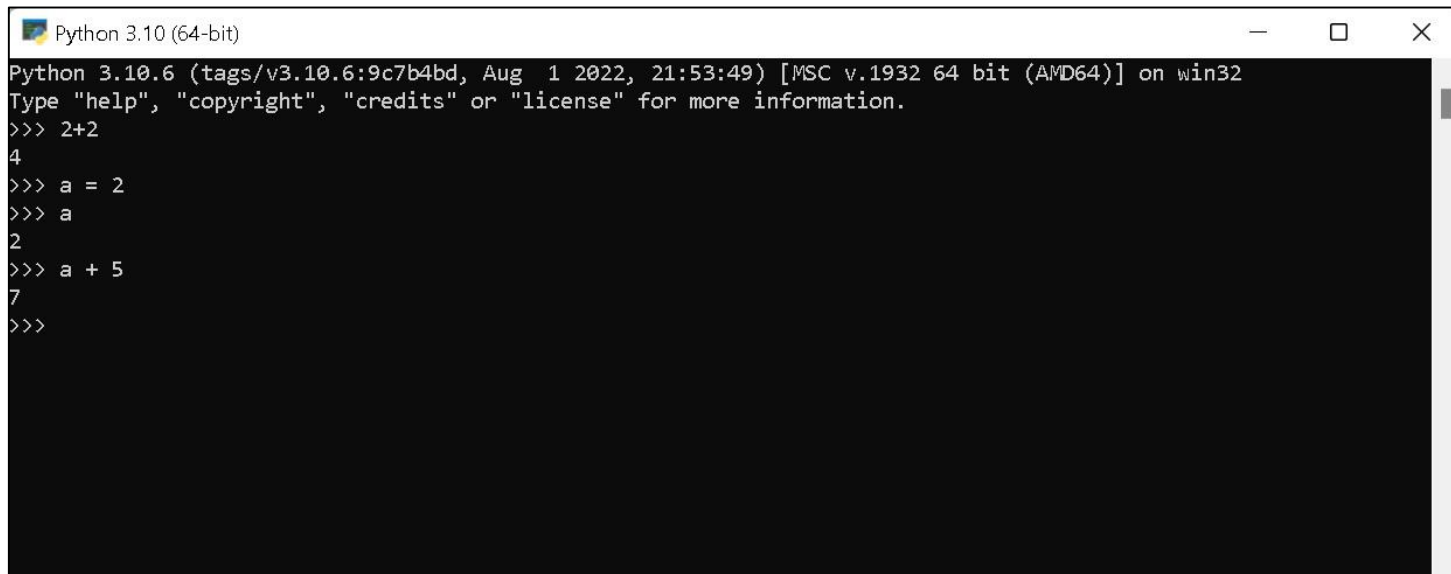
- ▶ Guido van Rossum – Dutch programmer – 1989
- ▶ General purpose, high-level programming language
- ▶ Main design consideration: readability
- ▶ Simple, easy to learn syntax
- ▶ Interpreter language, the written program can be run immediately
- ▶ Uses dynamic types and automatic memory management
- ▶ Platform independent
- ▶ Open source
- ▶ Object-oriented
- ▶ 3.11.5 (stable version, 24 August 2023)

Python programming language

- ▶ High level programming language
- ▶ Convert to a low-level language, since computers can only run programs.
- ▶ Portable
- ▶ The engine that converts and runs Python is called a Python shell.
- ▶ It can be used in two ways:
 - in interactive mode and
 - script mode.

Python programming language

- ▶ Typing Python expressions in the Python command line window in interactive mode – Python IDLE
- ▶ shows the result immediately

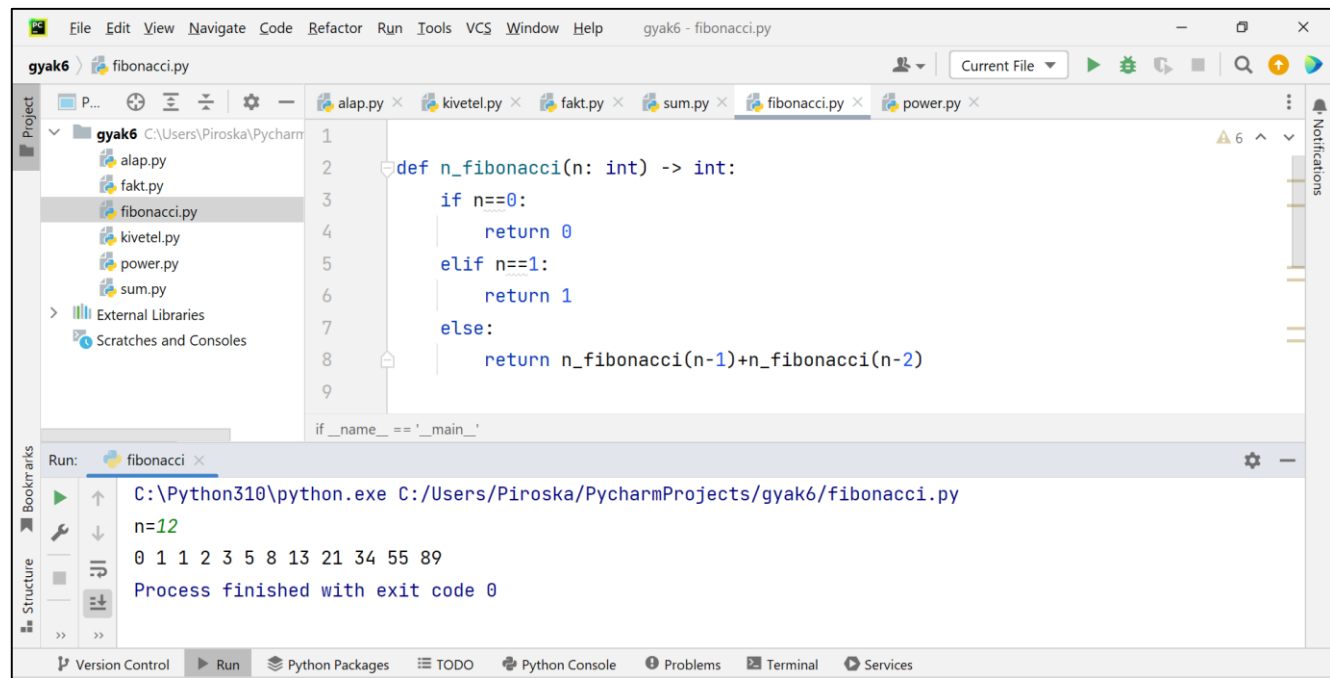
A screenshot of a Windows command prompt window titled "Python 3.10 (64-bit)". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The text inside the window shows the Python version and build information: "Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)] on win32". It then prompts the user to type "help", "copyright", "credits" or "license" for more information. The user has entered several commands: ">>> 2+2" which returns "4", ">>> a = 2", ">>> a" which returns "2", and ">>> a + 5" which returns "7". The prompt ">>>" is shown again at the bottom, indicating the interactive session is still active.

```
Python 3.10 (64-bit)
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> a = 2
>>> a
2
>>> a + 5
7
>>>
```

- ▶ **>>> Python prompt.**
- ▶ In PyCharm, the **Python Console** window allows you to use interactive mode.

Python programming language

- ▶ **Script** – write the program to a file and use the shell to execute the contents of the file.
- ▶ **PyCharm – Community Edition:**



Integrated Development Enviroment IDE

► Python

- **JetBrains PyCharm – Community version**
 - <https://www.jetbrains.com/pycharm/download/#section=windows>
- **IDLE:** the „official” Python developement enviroment
- **Visual Studio Code**
- **Thonny:** (most convenient for beginners)
- **Geany:** small, fast, programming–supporting text editor (recommended for Linux)
- Online development environments

Install

- ▶ **Python 3.11.5 – Installation instructions**

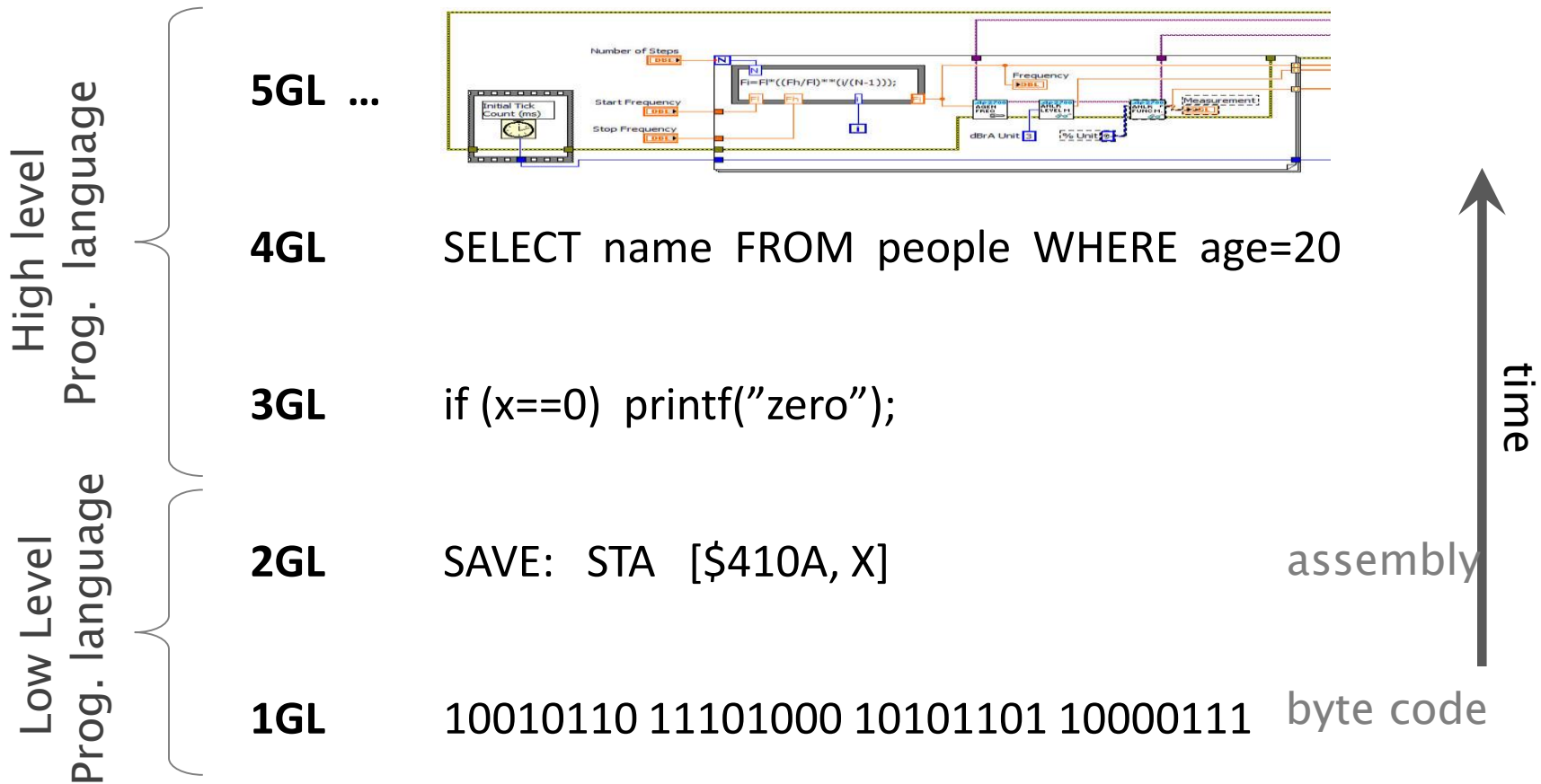
- https://www.youtube.com/watch?v=dAZb_6s_JIE&ab_channel=AmitThinks

- ▶ **Total Commander (optional)**

- ▶ **JetBrains PyCharm – Community version!**

- <https://www.jetbrains.com/pycharm/download/#section=windows>

The levels of programming



The levels of programming

First-Generation Languages: These are low-level languages like machine language.

Second-Generation Languages: These are low-level assembly languages used in kernels and hardware drives.

Third-Generation Languages: These are high-level languages like C, C++, Java, Visual Basic, and JavaScript.

Fourth Generation Languages:

These are languages that consist of statements that are similar to statements in the human language. These are used mainly in database programming and scripting. Examples of these languages include Perl, Python, Ruby, SQL, and MatLab.

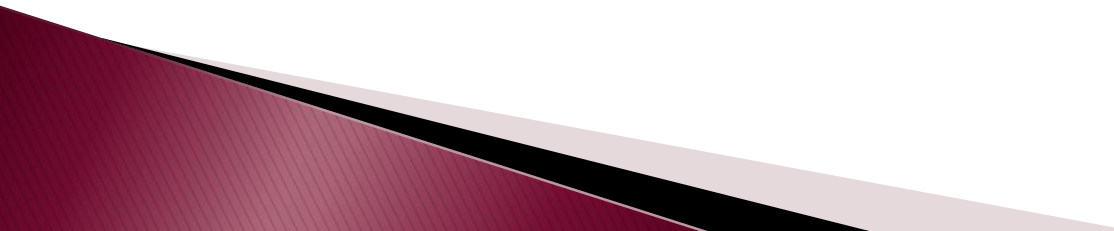
Fifth Generation Languages:

These are the programming languages that have visual tools to develop a program. Examples of fifth-generation languages include Mercury, OPS5, and Prolog.

Compiler and interpreter

- ▶ Processors only understand **machine code**
 - High-level code needs to be transformed
- ▶ Languages use different techniques:
 - **Compiler**
Pascal, C, C++, Labview
 - **Interpreter**
PHP, JavaScript
 - **Combined** (virtual machine bytecode)
Pl.: Java, C#

Source code → Executable code

- ▶ Source code
 - ▶ Syntactic rules
 - ▶ Semantic rules
 - ▶ Compiler
 - ▶ Interpreter
- 

Compiler

- ▶ A compiler is a special program that produces a **machine-code object program** from a source program written in high-level language.



- ▶ The compiler program treats the entire source program as a single unit and performs the following steps as it runs:
 - lexical analysis
 - syntactic analysis
 - semantic analysis
 - code generation

Interpreter

- ▶ Direct implementation
 - Analysis and code generation at runtime
- ▶ No object code
- ▶ Interpret instructions one by one
 - Single sequence of instructions as input
- ▶ Syntactically incorrect code can be executed
 - Errors remain hidden
- ▶ All runs require the interpreter
 - Interpretation and execution go together
- ▶ Execution is often slow



Compiler + Interpreter

- ▶ Some modern languages try to **combine** the two methods to get the best out of each (e.g. **Python**).
- ▶ When a source code is passed to Python, it first generates an intermediate code similar to machine code, called **bytecode**, which is then passed to an interpreter for execution.
- ▶ From a computer point of view, bytecode is very easy to interpret in machine language.
- ▶ This interpretation will therefore be much faster than the interpretation of a source code.

Program development, debugging

« debug »

- ▶ Programming is very complex and, like all human activities, we make many mistakes.
- ▶ For anecdotal reasons, programming errors are called **bugs**.
- ▶ The set of techniques used to detect and fix them is called the "**debug**".
- ▶ There are three types of bugs that can exist in a program:
 - Syntax errors (Syntactic)
 - Semantic errors
 - Run-time errors (Exceptions)

Syntax and semantics

- ▶ Syntax: formal rules for the text of a program.
- ▶ Semantics (logic): interpretation and meaning of language elements and algorithm.

Example (sign function):

```
x = int(input("x="))  
y = x
```

```
if x != 0:  
    if x > 0:  
        y = -x  
        print("Sign =", int(x/y))  
else:  
    printf("Sign =", int(x/y))
```

Semantic error ($x < 0$)



Syntax error (printf)



Run-time error ($x = 0$)



Syntax of programming languages

Fortran:

```
REAL FUNCTION FACT(N)
FACT=1
IF (N .EQ. 0 .OR. N .EQ. 1) RETURN
DO 20 K=2,N
20 FACT=FACT*N
RETURN
END
```

Python:

```
def Fact(N):
    f=1
    for i in range(1,N+1):
        f*=i
    return f
```

APL: $FACT \leftarrow \{ \times / 1 \omega \}$

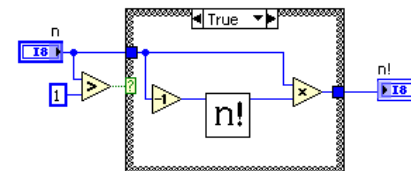
Pascal:

```
FUNCTION FACT(N:INTEGER):REAL;
BEGIN
    IF N=0 THEN FACT:=1
    ELSE FACT:=FACT(N-1)*N;
END;
```

C:

```
int Fact(unsigned N){
    return N<=1?1:N*Fact(N-1);
}
```

LabView:



Exercise

- Find syntactic and semantic errors in the following Python code, which raises **E** to the non-negative exponent **B**.

```
b = int(intpu("b="))  
r = 0  
  
wihle e < 0  
    r = r * b  
    e - 1 = e  
print("r =", r)
```


Solution

Syntax error:

intpu → input

wihle → while

Missing ":"

e-1=e → e=e-1

Run-time error:

Uninitialised e
Missing: input e

Semantic error:

Multiplicative unit element
 $r=0 \rightarrow r=1$

$e < 0 \rightarrow e > 0$

```
b = int(intpu("b="))
```

```
r = 0
```

```
wihle e < 0
```

```
    r = r * b
```

```
    e - 1 = e
```

```
print("r =", r)
```

Character set

- ▶ The **smallest components** of the source text of any program are characters.
- ▶ When compiling the source text, the character set is essential, and its elements can appear in the programs of a given language, and can be used to compile more complex language elements.
- ▶ For procedural languages, these are:
 - lexical units
 - syntactic units
 - Instructions
 - program units
 - compilation units
 - program

Character set

- ▶ Each language defines its own character set.
- ▶ There can be substantial differences between character sets, but programming languages generally categorise characters in the following way:
 - Letters (A–Z , a–z)
 - Numerals (0–9)
 - Other characters

Lexical units

- ▶ Lexical units are the elements of the program text that the compiler recognizes and tokenizes (intermediate form) during the lexical analysis.
- ▶ They are of the following types:
 - multi-character
 - symbol
 - symbolic name
 - tag
 - comment
 - literal

Identifier

- ▶ A sequence of characters that starts with a letter and can continue with a letter or a number.
- ▶ It is used by the programmer to name his own programming tools and then refer to them anywhere in the program text.
- ▶ **Default identifier:** A sequence of characters to which the language assigns a meaning, but this default can be changed or redefined by the programmer.
- ▶ This is usually the name of an implementation's tools (e.g. built-in functions).
- ▶ The standard identifier can be used in its original sense, but the programmer can also use it as his own identifier.

Keywords

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

Data types

- ▶ There are **statically** and **dynamically typed** programming languages.
- ▶ **Python** is **dynamically typed**, which means that variables do not have to be predefined, and the type of a given named variable can change during the program run.

Data types

- ▶ Python built-in types:
 - the primitives and
 - objects.
- ▶ **Primitive data** types denote a value. These are, for example, logical values and numbers.
- ▶ **Objects** are complex data types that usually combine other primitive types.

Primitive data types

bool

- ▶ Boolean, or logical value. There are exactly two values of this type, the logical True and the logical False.

int

- ▶ Integer number. These are positive or negative values of arbitrary size, without decimal places.

float

- ▶ Floating point number, or decimal. Values of this type are precise to a few decimal places and cannot store arbitrarily large integers.

NoneType

- ▶ In Python that represents the absence of a value. NoneType is a data type of the object when the object does not have any value. It suggests that a variable or function does not return a value or that a value is None or null. The None keyword is an object, a data type of the class NoneType. We can assign None to any variable, but we can not create other NoneType objects.

Objects

str

- ▶ String. Any text value of this type.

list

- ▶ List. It can store several different types of data, listed one after the other.

dict

- ▶ Dictionary, or associative array. It can also store several different types of data, but the data can be accessed by its own unique identification key.

tuple

- ▶ The elements of a tuple can be arbitrary Python objects. It is immutable (unmodifiable).

set

- ▶ Implements finite sets of unordered, unique elements. No indexing operators are defined for the type, but it is possible to iterate over its elements.

▶ **bytes**

- ▶ **complex types** – complex numbers.

Type conversion (Casting)

- ▶ **int**, **float** and **str** functions convert the parameters passed to them to **int**, **float** and **str** respectively.
- ▶ These functions are called **type conversion functions**.

Type conversion (Casting)

- ▶ The `int` function takes a real number or string as input and converts it to an integer value:

```
>>> int(3.9999) -> 3    #Return the whole/integer  
part of the number (truncate).  
>>> int(-3.999) -> -3  
>>> int('123') -> 123    #Convert a string into an  
integer.
```

Type conversion (Casting)

- ▶ The **float** type conversion function can convert an integer, a real number, and a syntactically correct string to a real number:

```
>>> float(17) -> 17.0
```

```
>>> float('123.45') -> 123.45
```

Type conversion (Casting)

- ▶ The **str** function converts its parameters to a string:

```
>>> str(17) -> '17'
```

```
>>> str(123.45) -> '123.45'
```

Variable

- ▶ the computer stores all data and instructions in binary form in memory.
- ▶ data is stored in variables.
- ▶ a variable is a name that contains a stored, remembered value or partial result.
- ▶ a variable can be assigned a value using a value assignment expression.
- ▶ variable names can be of any length, can contain letters and digits, but must start with a letter or underscore character.

Variable

- ▶ A variable is a programming tool that has four components:
 - name (identifier)
 - attributes (type)
 - address (memory)
 - value
- ▶ In Python, it is not necessary to free the memory occupied by the variable/object, as it will automatically take care of freeing it.

Variable

- ▶ The **name** is an identifier. In the program text, the variable always appears with its name, which can be any component.
- ▶ You can look at things by assigning the other three components to the name.
- ▶ **Attributes** are characteristics that determine the behavior of a variable at runtime.
- ▶ In procedural languages (usually typed languages), the main attribute is the type, which delimits the range of values that a variable can take.
- ▶ Attributes are assigned to variables by declaration.

Variable

- ▶ There are different types of declaration.
- ▶ **Explicit declaration.** The attributes must be specified for the full name of the variable. Languages usually allow to assign the same attributes to several variable names at the same time.
- ▶ **Implicit declaration:** done by the programmer, assigning attributes to letters in a separate declaration statement. If a variable name is not in an explicit declaration statement, the variable will have attributes assigned to the initial letter of its name, so variables with the same initial letter will have the same attributes.

Variable

- ▶ **Automatic declaration:** the compiler assigns an attribute to variables that are not explicitly declared and do not have an attribute assigned to their initial letter in an implicit declaration statement. Attribute assignment is based on a character (often the first) of the name.

```
>>> a = 2
>>> a
2
>>> type(a)
<class 'int'>
```

Variables – Assignment

```
>>> b = 2.5
```

```
>>> b
```

```
2.5
```

```
>>> type(b)
```

```
<class 'float'>
```

```
>>> c = 'a'
```

```
>>> c
```

```
'a'
```

```
>>> type(c)
```

```
<class 'str'>
```



Variables – Assignment

```
>>> a = 2  
>>> d = a + 2.5  
>>> type(d)  
<class 'float'>
```

```
>>> a = b = c = 1 #multiple assignment  
>>> a  
1  
>>> b  
1  
>>> c  
1
```

Variables – Assignment

```
>>> a , b, c = 5, 2.3, 'apple'  #multiple assignment
>>> a
5
>>> type(a)
<class 'int'>
>>> b
2.3
>>> type(b)
<class 'float'>
>>> c
'apple'
>>> type(c)
<class 'str'>
```

Variables – Assignment

```
>>> x = 18
```

```
>>> y = 9.2
```

```
>>> z = x + y  # is correct because the variables on  
the right side of the assignment have values.
```

```
>>> x = x + 1  # increase the value of x by 1
```

```
>>> a = 'table'
```

```
>>> b = 'tennis'
```

```
>>> c = a + b
```

```
>>> c
```

```
'tabletennis'
```

Variables – Assignment

```
>>> l = 20
```

```
>>> w = 5 * 9
```

```
>>> l * w
```

```
900
```

```
>>> type(l) # type of the variable  
<class 'int'>
```

```
>>> (x,y)=(10,20) #tuple (later)
```

```
>>> x
```

```
10
```

```
>>> y
```

```
20
```


Constans (literals)

- ▶ variables whose value is **fix** are called constants.
- ▶ **Examples:**
 - 5 (int)
 - 5.0 (float)
 - 5.2 (float)
 - 52E-1 = 5.2 (float)
 - "Hello" (string)
- ▶ One of the weaknesses of the language is that there is no real constant type, by convention variables written in all upper case are considered "constants".
- ▶ We must be careful to keep the contents of variables declared in this way intact throughout the execution of the program.

Operators

- ▶ addition: +
- ▶ subtraction: -
- ▶ multiplication: *
- ▶ division: /
(result is always a float; e.g. $8/4 = 2.0$)
- ▶ integer division: //
(pl. $7//3 = 2$; $-7//3 = -3$)
- ▶ remainder: %
(result is always a int; e.g. $7\%3 = 1$)
- ▶ exponentiation: **
(pl. $2**3 = 8$)

Operators precedence

- ▶ The bracket has the greatest precedent.
- ▶ Multiplication is the second most powerful operation.
- ▶ Multiplication and division are operations of equal strength.
- ▶ They have a higher precedence than addition and subtraction, which are also of equal strength.
- ▶ Equal strength operators are evaluated from left to right.

Operators – Examples

```
>>> 2 + 2
```

```
4
```

```
>>> 50 - 5*6
```

```
20
```

```
>>> (50-5*6) / 4
```

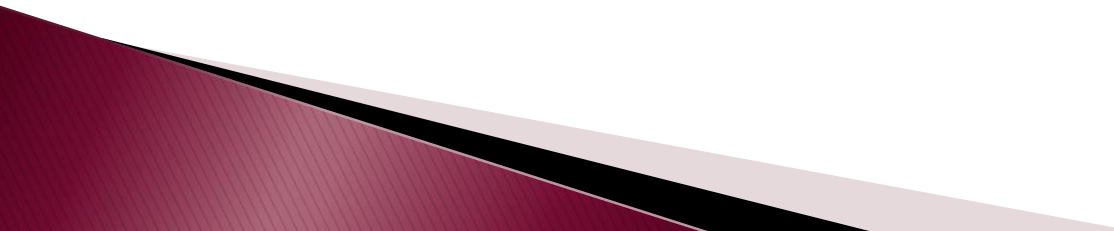
```
5.0
```

```
>>> 8/5
```

```
1.6
```

```
>>> 17/3
```

```
5.666666666666667
```



Operators – Examples

```
>>> 17 // 3
```

```
5
```

```
>>> 17 % 3
```

```
2
```

```
>>> 5 * 3 + 2
```

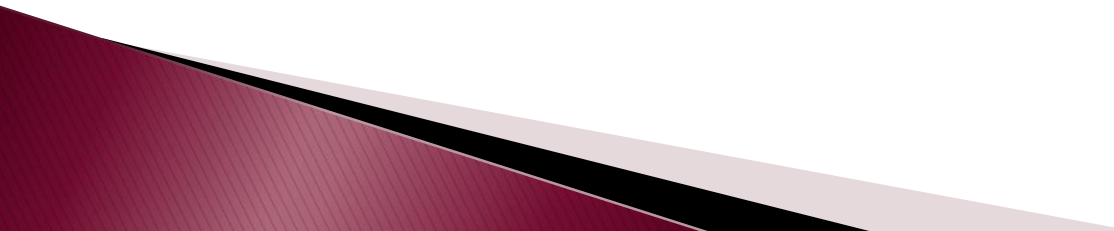
```
17
```

```
>>> 5 ** 2
```

```
25
```

```
>>> 2 ** 7
```

```
128
```



'+' operator

- ▶ a + operator for strings indicates the concatenation operation, which is used to **concatenate two strings**.
- ▶ For example:
 - ▶ >>> s1 = 'apple',
 - ▶ >>> s2 = ' tree'
 - ▶ >>> s1 + s2
 - ▶ apple tree
 - ▶ The result of the concatenation is the apple tree.
 - ▶ The space before the ' tree' is also part of the string, because there must be a space between the two words in the result.

'*' operator

- ▶ The * also works for strings, as a sign of a **repeat operation**.
- ▶ For example, 'Fun'*3 results in 'FunFunFunFun'.
- ▶ One operand of the operation must be a string, the other an integer.
- ▶ >>> 'Fun'*3,
FunFunFunFun
- ▶ In some ways, the interpretation of + and * as seen above is analogous to the mathematical interpretation.
- ▶ $4*3$ is equal to $4+4+4$, so in the case of 'Fun'*3 we can expect to get 'Fun'+ 'Fun'+ 'Fun', and it is.
- ▶ >>> 'Fun'*3
'FunFunFun'
- ▶ >>> 'Fun'+ 'Fun'+ 'Fun'
'FunFunFun'

Comment

- ▶ An annotation is a programming tool that allows you to place a sequence of characters in a program that is not addressed to the compiler, but to the human reading the program text.
- ▶ It is **an explanatory text** that helps the user to use the program, giving information about how it works, the circumstances in which it was written, the algorithm used, the solutions adopted. The commentary is **ignored or omitted by the compiler** during the lexical analysis.
- ▶ Any character in the character set may be used in the comment, and all characters are equivalent, representing only themselves, and character categories are irrelevant.
- ▶ The following options are available for placing a comment in source text:
 - ▶ # (at the beginning or between lines)

Triple quotation mark

- ▶ To make it easier to insert special characters into a string without using the backslash or being able to insert the backslash itself, the string can be delimited by a triple apostrophe or a triple quotation mark:

```
>>> print("""Don't be lazy.""")  
Don't be lazy.
```

- ▶ We can also format the printout.

```
>>> print("""First line  
... Second line  
... Third line""")  
First line  
Second line  
Third line
```

Expressions

- ▶ expressions consist of **operands** and **operators**
- ▶ **operands** provide the data: variables or specific values, constants
- ▶ **operators**: operation symbols: $+$, $-$, $*$, $/$ /expressions can be nested
- ▶ in the case of several operators, the precedence of the operators determines the order of evaluation
- ▶ the order can be explicitly defined by bracketing

Escape sequences

- ▶ `\a` – ASCII backspace
- ▶ `\t` – ASCII horizontal tab (TAB)
- ▶ `\r` – ASCII carriage return (CR)
- ▶ `\n` – ASCII linefeed, new-line (LF)
- ▶ `\\` – display back-slash
- ▶ `\'` – display apostrophe
- ▶ `\"` – display quotation mark

Output

- ▶ **print() function**
- ▶ `print()`, after having printed its parameters, will, by default, move the cursor to the beginning of the next line.
- ▶ If you don't want it to start a new line at the end of the printout, you need to override it.
- ▶ The override can be done by setting an optional parameter to the function called `end`.
- ▶ `print("2+3 =", 2+3, end="")`. Since the `end` parameter is set to an empty string, the next printout will start after the result.

Output – Examples

```
>>> print('Hello')
```

Hello

```
>>> print('apple' + 'tree')
```

appletree

```
>>> print('apple' + '\n' + 'tree')
```

apple

tree

```
>>> print('I don\'t want to be lazy.')
```

I don't want to be lazy.

```
>>> print("I don't want to be lazy.")
```

I don't want to be lazy.

Input

- ▶ When input or read data, we ask the user for information.
- ▶ For this purpose, we use the `input()` function in Python.
- ▶ The `input()` function always reads a string.
- ▶ The function can be given a text as a parameter, which will be displayed in the result window and will indicate what the program is waiting for.

```
x = input("Please enter the year of birth:")
```

```
year = int(x)
```

- ▶ Nested within each other:
- ▶ `year= int(input("Please enter the year of birth:"))`

Input – Examples

```
>>> name = input()
Alex
>>> name
'Alex'
>>> type(name)
<class 'str'>
>>> name = input('Name=')
name=Alex
>>> name
'Alex'
>>> year=int(input('Year='))
Year=18
>>> type(year)
<class 'int'>
```

Exercise

- ▶ Write a Python program that reverses/swap two values.

Solution

```
a = int(input("a="))  
b = int(input("b="))
```

#swap1 - auxiliar variable

```
c = a  
a = b  
b = c  
print(a, b)
```

#swap2 - arit. operators

```
a = a + b  
b = a - b  
a = a - b  
print(a, b)
```

#swap3 - arit. operators

```
a = a * b  
b = a // b  
a = a // b  
print(a, b)
```

#swap4 - bitwise exclusive or

```
a = a ^ b  
b = a ^ b  
a = a ^ b  
print(a, b)
```

#swap5 - tuple

```
(a,b)=(b,a)  
print(a, b)
```

Functions in the Math module

- ▶ `floor(x)` – round `x` down
- ▶ `ceil(x)` – round `x` up
- ▶ `sqrt(x)` – square root of `x` (same as `x ** 0.5`)
- ▶ `pow(x,y)` – raise `x` to the power of `y` (same as `x ** y`)
- ▶ `gcd(a, b)` – greatest common divisor of integers `a` and `b`
- ▶ `factorial(x)` – `x` is factorial (`x` is an integer)
- ▶ `log(x)` – natural logarithm of `x`
- ▶ `pi` – constant `pi = 3.1415926`
- ▶ `sin(x)` – sine of angle `x` (`x` is given in radians)
- ▶ `cos(x)` – cosine of angle `x` (`x` must be given in radians)
- ▶ `tan(x)` – tangent of angle `x` (`x` must be given in radians)
- ▶ `degrees(x)` – angle `x` in degrees given in radians
- ▶ `radians(x)` – angle `x` given in degrees in radians

Homework

Exercise 1.

- ▶ Installing software
- ▶ Learning and practising Math module functions

Exercise 2.

- ▶ Input the radius of a circle, the program have to calculate its circumference and area (use the PI constant defined in the `math` module: `math.pi`).

Exercise 3.

- ▶ Input two integers, write a program which calculate the arithmetic and geometric mean of the numbers.