Basic concepts of Python language

- Data types and values
- Expressions and statements
- Flow control and functions
- Data structures: lists, tuples, sets, dictionaries
- Basic input and output
Basic data types

- Examples:
  - int: 12, 0, -2
  - float: 1.02, -2.4e2, 1.5e-3
  - complex: 3+4j
  - bool: True, False
  - string: "Test string"
Conversion between types

- `int(-2.8) → -2`
- `float(2) → 2.0`
- `int("123") → 123`
- `bool(-2) → True, bool(0) → False`
- `str(234) → "234"`
Strings 1

- A sequence of characters
- delimited between single (’) or double (”’) quotes
- This optionality is useful if a string needs to contain a quotation mark. You can also achieve this by escaping the quotation mark: \" or \’
- Can also contain escape sequences like \n for newline and \t for tabulator. See page 41 of the course page for all escape sequences.
- Strings can be concatenated with the + operator: "compu" + "ter" results a new string "computer".
If you want a string to contain several lines of text, use triple quoted strings:
"""First line
second line"""
Expressions 1

- An *expression* is a piece of Python code that results in a value
- It consists of values combined together with operators
- Values can be *literals* or variables
- Operators include arithmetic operators, comparison operators, function calls, indexing, attribute references, among others.
- Examples of expressions:
  - $1+2$
  - $c > 0$ and $c != 1$
  - $7/(2+0.1)$
  - $(1,2,3)$
  - $a$
  - $a < 5$
  - $\cos(0)$
  - $\text{obj.attr}$
  - $\text{mylist}[1]$
  - $(-1)**2 == 1$
Expressions 2

- Usually expressions don't do anything beyond resulting a value, but sometimes an expression, typically a function call, can have *side-effects*.

- Side-effects that functions may have, should be well documented. Otherwise unexpected behaviour and confusion can occur, and the reason for this is hard to pinpoint.

- For example, the function (method) `writeline` has the side-effect of changing the contents of a file.

- Programming without any side-effects is called *functional programming*.

- Python is not a functional programming language, it is an *imperative language*. 
Statements 1

- **Statements** are commands that have some effect
- Simplest example of statements is just an expression:
  - Usually these are function calls
  - Are also used in interactive use: the interpreter prints the computed value of an expression
- The print statement prints the values of expressions: `print expr1, expr2, ...`
- Another simple statement is an assignment: `variable_name = expression`
Variables

- Variable is a name that refers to a value or an object
- With the assignment operator, you can bind a variable name to a value, and rebind it later to another value
- In Python types are not attached to variable names; types are attached to objects/values in the memory of the computer
  - This is runtime/dynamic typing
- Therefore a variable name var can at one time point to a string and a few seconds later point to a complex number
- This can make your program a bit confusing: try to avoid it
Bindings and type checking in C and Python

C – static typing

Python – dynamic typing

variable_name

memory

int

str