

# Programming 1

Introduction in  
programming



**Marian Neagul**



**Teodora Selea**



**Elena Flondor**



**Adrian Spătaru**



**Florin Roșu**



**Cosmin Bonchiș**

# What we will talk about?

- Course information
  - Class requirements and evaluation
- Basic elements about Python
  - Variables and data types
- Mathematical operations

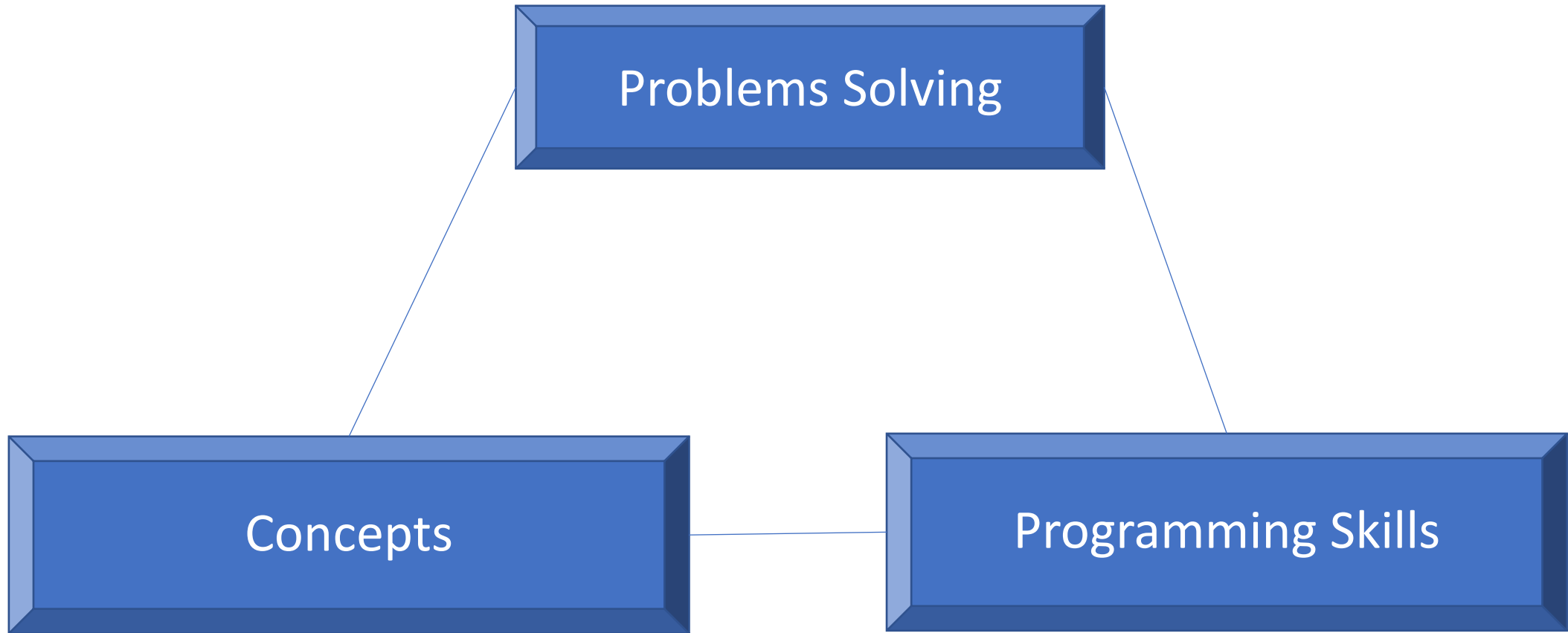
# Course Information

- An introductory course
- For beginners
  - You cannot learn programming in a passive way
  - Do not be afraid of Python. Failure is an opportunity to learn
  - Download our lecture and laboratory materials
  - Use any additional learning source that is suitable for your needs
  - Exercise

# Evaluation

- 40% Examen
- 60% Laboratory (4 Tests)
  
- 1 Bonus point for activity

# About the course



# Course content

- What is a computer and how is functioning
- Knowledge representation using data structures
- Iterative/recursive programming
- Algorithms verification
- System organization using modules, classes and objects
- Guidelines for writing programs
  - Simple principles
  - Refactoring

# What does a computer?

- Fundamentally
  - Runs millions of **computations** per second
  - Stores the result of the computations
    - Uses hundreds of giga of memory
- Computations types?
  - Computations **predefined** in language
  - Computations **defined by us** - programmers
- Computers do/know what we tell them, nothing else



# Types of knowledge

## Declarative knowledge

- Assertion regarding facts
- Example
  - $\sqrt{x}$  is defined to be a value  $y$  for which  $y^2 = x$  and  $y > 0$
- The definition above is axiomatic. But this definition does not help, in general, to find the square it tells how to test a value
- It tells **WHAT** but **NOT**

## Imperative Knowledge

- Recipe. A set of rules.
- Tells how to deduce something
- Example: square root

# Types of knowledge

## Heron of Alexandria Method

Choose random a number  $G$

If  $G \cdot G$  is close enough to  $x$ :  
then stop and the answer is

Otherways:

guess another  $G$ , based on the following  
formula  $G = (G + x/G) / 2$

Repeat

- Opposite to declarative knowledge here we have a method that tells what to do in order to solve the problem
- It is Clear!

# Types of knowledge

## Heron of Alexandria Method

For  $x=9$

$g$	$g * g$	$x/g$	$(g+x/g)/2$
2	4	4.5	3,25
3,25	10,5625	2,7692	3,0096
3,0096	9,0576	2,9904	3

**Fast?**

# Recipe

1. A sequence of **steps**
2. A mechanism to **manage the control**, that tells when each step must be executed
3. A modality to tell **when to stop**

**When all three characteristics are satisfied we can talk about an ALGORITHM**

# First computers – Numeric computers with fix programs

- Office calculators
  - Just for arithmetic operations
- [Atanasoff–Berry](#) Calculators (1942)
  - Solve linear equations
- Turing Machine
  - an electro-mechanical device used to decode Enigma messages, a German device for encoding message in WW2

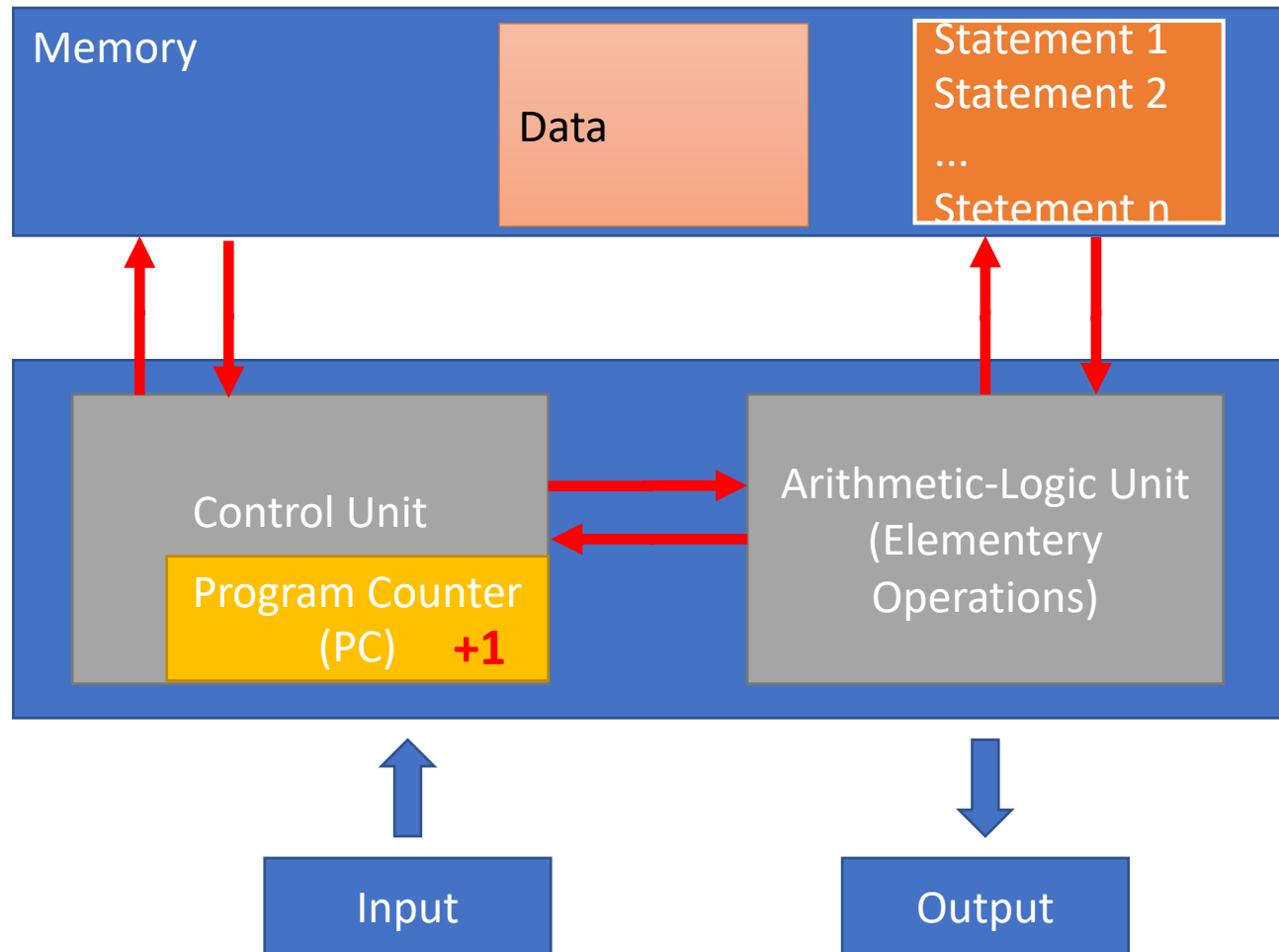


# First computers – Numeric computers with fix programs

- Could we create a computer with a fix program that receives like input a diagram of a circuit of an another computer and configures in such a way it works like it is described in that? 🤔
  - Could it behave like an office calculator or like an Atanasoff-Berry calculator?

Exists! It is the hard of each computer: an *interpreter*.

# First Computers – Numeric computers with a stored programs



## Elementary architecture

- Some statements can modify PC to go to a specific statement
  - Flux controller

# Numeric computers with a stored programs

- A sequence of statements stored into computer memory
  - A predefined set of primitive statements
    - Arithmetic and Logic
    - Simple conditional statements
    - Data copy
- An interpreter (a special program) which executes the statements in an order
  - Uses conditional statements to control the statements flow
  - Stops when it is finishing



# What is a program?

A program is a recipe!

- It is formed from a fix set of statements and primitives
  - With this set anything can be
- Which are this primitives?
  - move left, move right, read, write, scan, noop
- In 1936 Alan Turing demonstrated that 6 primitives are enough to describe any program that can be described through a mechanic process

One implication of the previous statement is that a program written into a language can be translated in any other language.

Concept also known like “Turing Compatibility”.

# What is a program?

- To describe/present recipe it is necessary a language. A programming language. This course uses Python like programming language!



This is not a course about Python Language!

# Recipe (algorithm)

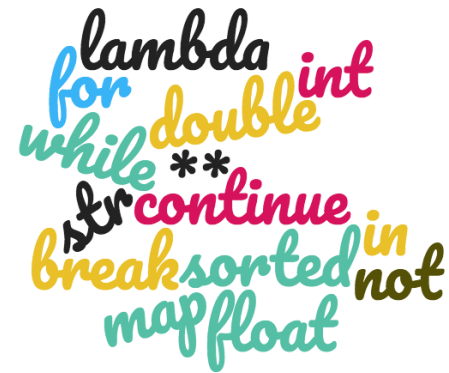
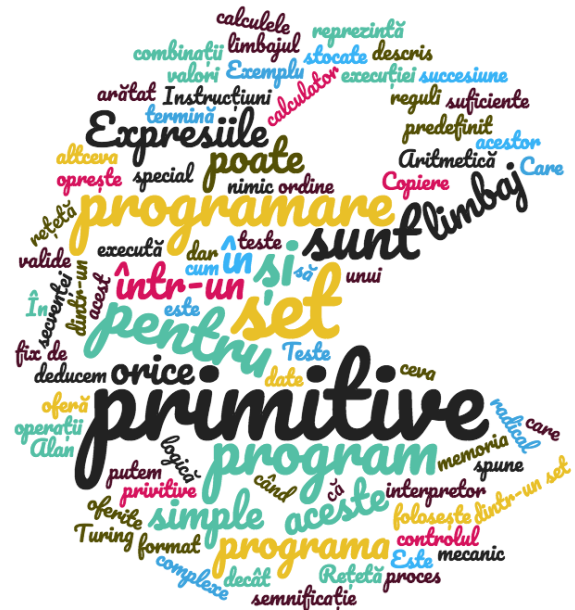
- A programming language is formed from a set of primitive operations
- **Expressions** are complex (but valid) combinations of programming language primitives
- The expressions and calculus have values and meaning into a programming language

# Languages

- Primitive statements

  - Linguistic: words

  - Programming language: numbers, strings, simple operators



# Programming languages

Low/High Level ?

C

Python

JavaScript

Go

For general Usage  
or Dedicated?

Matlab

Python

JavaScript

Compiled or  
Interpreted?

C

Java

Python

JavaScript

# Programming languages

- **Syntax**

- Which expressions are linked to this programming?
- „Boy a cat in the house” 🤔

- **Semantic**

- **Static**: tell which program have sense. What expression have sense?
  - „This course is tasty” .
  - Syntactically right?
  - This kind of errors can generate unexpected behavior...
- **Complete**: what is the expected result? What happens when you execute the program?
  - ✓ **STYLE**: Depends on programmers!
  - ✓ **Result**: fails, never stops, stops with an unexpected result or stops with an expected result

# Objects (in Python)

- Programs use objects
- Objects have a type that tells the programming language what is possible to do with this objects
  - Moris is a dog and can do „hau hau”
  - Pisi is a cat and can do „miau miau”
- Objects can be
  - Scalar values (ex. value 42)
  - Non-scalar (have an internal structure that can be accesed)

# Scalar values

- Fundamental data types
  - Numbers
    - 3 – int
    - 3.14 – float
  - Booleans
    - True/False – bool
  - NoneType
    - Only one value: None
  - Strings
    - 'Timișoara' – str
- Usually we talk about the pair (type, value)
- For each data type we have a list of operators
  - For Numbers: + - \* / %
  - For Strings: +, \*

Can use type() to find an object type:

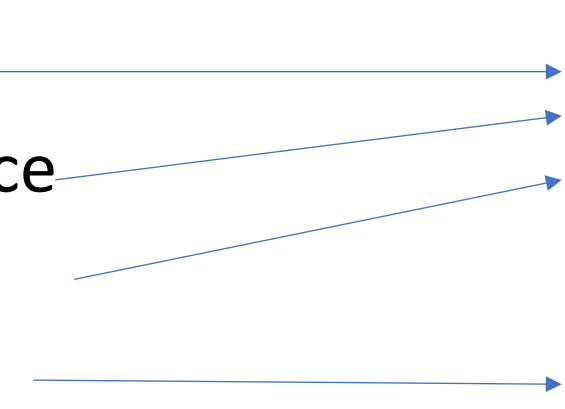
```
>> type(5)
Int
>> type(3.14)
float
```



# Data Type Conversion

- Can convert objects of one type into another type
  - `float(3) → 3.0`
  - `int(3.9) → 3`

# Operators for `int` and `float`

- $i+j$  → addition
  - $i-j$  → difference
  - $i*j$  → multiply
  - $i/j$  → divide
- If both operands are int the result is int  
If one operand is float the result is float
- Result of type float
- $i\%j$  → modulo  $i$  at  $j$
  - $i**j$  →  $i$  at power  $j$
- 

# Operators

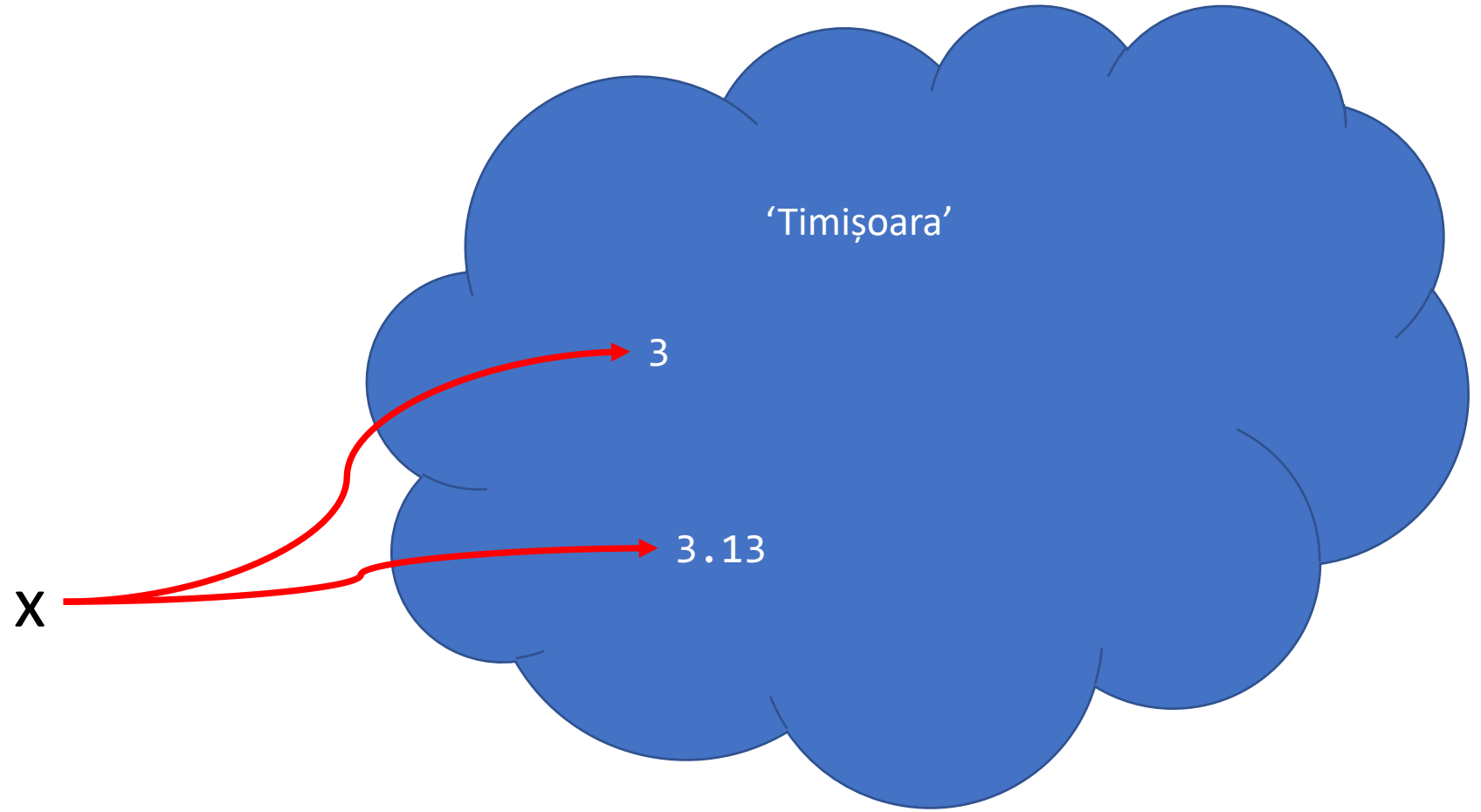
Operator	Descriere
**	Power
~ + -	Complement, plus și minus unary
* / % //	Multiply, divide, modulo și „floor division”
+ -	Addition și subtraction
>> <<	Byte shifting(at right, at left)
&	<b>AND</b> on bytes
^	<b>EXCLUSIVE OR</b> on byte, <b>OR</b> on byte
<= < > >=	Relational operators
<> == !=	Equal operators
= %= /= //= -= += *= **=	Assignment/short operations operators
is is not	Identity operators
in not in	Membership operators
not or and	Logic operators

# Variables

- Creates an association/link between a name and a value
  - Assignment operation: `x=3`,
    - Creates an association/link between value 3 and name x
- What is the type of a variable?
  - For Python: the type is inherited (identified) from the value linked to the variable. The variable does not have a type, the value referred by it has
  - The variable type is dynamic, it changes based on the referred value
  - Advice: do not change randomly the variables data type
- Variables names: it is important to have a since!
  - Do not use reserved words

# Variabiles

$x = 3$



# Expressions

- Way to name an expression?
  - To reuse the name later!
- Can change the code easily later

```
pi = 3.14159
```

```
radius = 2.2
```

```
area = pi * (radius ** 2)
```

# Strings

- Letters, numbers, special characters, spaces
- Defined using the symbols `"` or `'`  
`hi = 'salut'`
- Concatenation  
`name = 'ion'`  
`salut = hi + name`
- Other operations  
`test = hi + " " + name * 2`

# Input/Output: print

- Used to display at standard output
- print keyword

```
x = 1
```

```
print(x)
```

```
x_str = str(x)
```

```
print ("A number is", x, ".", "x=", x)
```

```
print ("A number is" + x_str + ". " + "x= " +  
x_str)
```



# Input/output: input("")

- Displays what it receives like argument (what is specified between quotation marks)
- Reads what the user is typing until it encounters the key ENTER
- Returns a value that is associated/linked to a variable

```
text = input("Type something: ")
```

```
print(5*text) 🤔
```

- input returns an string that has to be converted at desired data type

```
num = int(input("Type an integer number"))
```

```
print(5*num)
```

# Comparing operator for int, float, str data type

- *i* and *j* variables names
- All below comparison are evaluated to boolean values

*i* > *j*

*i* >= *j*

*i* < *j*

*i* <= *j*

*i* == *j* → equal operator, True if the value *i* is equal with the value of *j*

*i* != *j* → different operator, True if the value of *i* is different from the value of *j*

*i* is *j* → identity operator, True if *i* is the same with *j*

# Logic operators

- a and b are boolean variables

not a            →     True if a is False  
                              False if a is True

a and b         →     True if a and b are True

a or b          →     True if any a and b is True

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

# Conditional statement

```
if <condition>:  
    < expression >  
    < expression >  
    ...
```

```
if < condition >:  
    < expression >  
    < expression >  
    ...  
else:  
    < expression >  
    < expression >  
    ...
```

```
if < condition >:  
    < expression >  
    < expression >  
    ...  
elif < condition >:  
    < expression >  
    < expression >  
    ...  
else:  
    < expression >  
    < expression >  
    ...
```

- <conditie> an expresion that evaluates to a boolean value True or False

# Alignment

- VERY important in Python
- Used to identify blocks of code

```
x = float(input("x="))
y = float(input("y="))
if x == y:
    print("x and y are equal")
    if y != 0:
        print ("so, x/y is", x/y)
elif x<y:
    print ("x is smaller")
else:
    print("y is smaller")
print ("END PROGRAM")
```

# Bibliography

- <https://youtu.be/0jljZRnHwOI?t=1020>
- [John Zelle](#), **Python Programming: An Introduction to Computer Science (chapter 1)**