Basics

Spaces and indentation (tabs) are relevant parts of the code

Instructions in a *block of code* will have the same indentation

A *block of code* contains one or more lines of code inside it. The contained lines will be indented one more level than the container one.

A colon **:** opens a new *block of code* in the following line.

at the beginning of a line marks this line as non-executable. For example

 # This is a single line comment

To create multiple line comments, use three apostrophees in a row

'''

This is a

Multiline comment

'''

Variables

There's no declaration of variables. When you use a new name in an assign sentence, that becomes a variable of the type of the value assigned to it	**a = 10** creates a variable called *a* that stores an *integer* value
The decimal separator is the point	**a = 3.14** creates a variable called *a* that stores a decimal (*float*) value
To transform a numerical value into a string you *cast* it using the function **str()**	**str(10)** will create the string value **"10"**
To transform a string value into an integer you *cast* it using the function **int()**	**int("10")** will create the integer value **10**

A variable contains a literal value of a certain type (integer, alphanumeric, decimal, boolean, etc) and it can be used to perform different operations or construct logical expressions

Basic input / output

print(a)	Prints the content of the variable called *a* and opens a new line	**print**("Helo world") **Hello word** and opens a new line
print(a,end=",")	Prints the content of the variable called *a* and then a comma without opening a new line	**print**(name,end=",") **My name,**
input()	Reads a **string** value form the keyboard	**a = input()** will store in a variable called *a* the value entered by the user as a **string**
int(input())	Reads a **string** value form the keyboard, ant *casts* it into an **integer**	**a = int(input())** will store in a variable called *a* the value entered by the user casted as an **integer**

Arithmetic operators

+	add	12 + 5 returns 17
-	subtract	12 - 5 returns 7
*	product	12 * 5 returns 60

Arithmetic operators (cont)

/	decimal division	12 / 5 returns 2.4
//	division (whole numbers)	12 // 5 returns 2
%	remainder of the division	12 % 5 returns 2
**	exponentiation	12 ** 5 returns 248832

Comparison operators (logical)

<	less than	12 < 5 evaluates as **False**
<=	less than or equal to	12 <= 5 evaluates as **False**
==	equal to	12 == 5 evaluates as **False**
>=	greater than or equal to	12 >= 5 evaluates as **True**
>	greater than	12 > 5 evaluates as **True**
!=	not equal to	12 != 5 evaluates as **True**

In a comparison, the sign of equality (=) can never be alone as it would be confused with the assignment of values (*a = 10*). This is why the logical equality operator is a double sign of equality. Therefore **a=10** means *assign the value 10 to the variable a* and **a==10** means *is the content of the variable a a number 10?*

Logical operators

<expr1> **and** <expr2>	**True** if and only if the two expressions are **True**	(a>0) **and** (a<5)
<expr1> **or** <expr2>	**True** if and only if at least one of the two expressions is **True**	(a<0) **or** (a>=5)
not <expr>	**True** if and only if <expression> is **False**	**not** (a==0)

By *logical* we understand an expression or operation that can only take two different values: **True** or **False**

Lists

len	Calculates the number of elements in the list.	while (position < len(myList)):
update	Modifies the content of a position of the list.	myList[position] = newValue
append	Add a new position at the end of the list, and stores there a value.	myList.append(newValue)
insert	Creates a new position in the middle of the list, and stores there a value.	myList.insert(position, newValue)
pop	Retrieves the value stored at a given position, and then, removes the position.	value = myList.pop(position)
delete	Erases a position of the list.	delete(myList[position])
remove	Locates the first occurrence of a value in the list, and then erases its position.	myList.remove(value)
in	Membership operation. Check for the existence of a value in the list.	if (value in myList):
index	Locates the first occurrence of a value in the list, and returns it's position. In case the valoue is not found, it will generate a run-time error.	position = myList.index(value)
count	Counts the number of occurrencies of a value in the list.	occ = myList.count(value)

Lists (cont)

reverse	Flips all the values in the list, so the first will become the last and viceversa.	myList.remove()

Maths

abs(*arg*)	receives an **integer** number as an argument and returns the integer absolute value	**abs(-12)** returns 12
math.fabs(*arg*)	receives a **float** as an argument and returns the **float** absolute value	**math.fabs(-12.34)** returns 12.34 **math.fabs(-12)** returns 12.0
math.floor(*arg*)	receives a **float** as an argument and rounds it down to the nearest **integer**	**math.floor(2.5)** returns 2 **math.floor(-3.4)** returns -4
math.ceil(*arg*)	receives a **float** as an argument and rounds it up to the nearest **integer**	**math.ceil(2.5)** returns 3 **math.ceil(-3.4)** returns -3
math.pi	returns the value of Pi	**math.pi** returns 3.141592653589793

You will need to **import math**

Flow control (more to be added along the course)

if (*<expr>*) **:**	Forks the execution stream according to the logical value of the expression *<expr>*	**if** (a == b)**:** <do something>
else:	As part of an *if .. else* block, starts the block code to be executed if the *expresion* was **False**	**else:** <do something esle>
elif (*<expr2>*)**:**	Compound an *else:* statement with a new *if* statement	**elif** (a<b)**:** <and another>
while (*<expr>*)**:**	Generates a *loop* that will run as long as the expression *<expr>* is **True**	**while** (a<10)**:** <do something> <update *a*>