Basic data and operations.

Variables and conditionals.

Olatz Perez de Viñaspre Garralda
Ander Soraluze Irureta

Help

Python's documentation:
https://docs.python.org/3/

GOOGLE in general!!!

Talking with the computer

Print a message

```
print(“message”)
```

Let’s try it

02-01-Hello_world.py

In short...

- Do not be afraid of trying
- Errors are part of the development:
  - They are useful for learning
- SyntaxError: we wrote something wrong
  - Give a look to parenthesis, quotes, tabulators,...
- NameError: we wrote the name wrong
  - Check documentation for the correct name of the function

Secret messages?

You can add **comments** to your code that will not be executed

```
#The following code will print “message”
print(“message”)
#And the following “hello”
print(“hello”)
```

Strings

They are written between quotes

For example,

> **“Amaia”**
>
> **‘Manuel Lardizabal Pasealekua’**

Python

There is no difference between simple or double quotes

“Message” == ‘Message’

Be careful! Use the same for opening and closing

“Message’ **ERROR**

Variables

They store the data

There are different data types:

- Integer numbers: int
- Float numbers: float
- Strings: str
- Boolean: True or False

Variable. Previous example with variables:

```
message = “Hello world”
print(message)
```

Variables in Python

- We use the = symbol to assign a value
  - The left side takes the right's value
  - variable = value
- Name convention, first character lowercase
  - Mandatory, fist character alphabetic (no numeric)

Talking with the computer

Read/store a value

```
name = input(“What’s your name?”)
```

Let’s try it

02-02-Hello_name.py

In short...

- Print can print several values
  - print(“Hello”, name)
- Strings can be concatenated
  - message = “Hello” + ” world”

Remember...

- print displays the VALUE of a variable
  - print(**name**)

Numbers

The value is used/assigned directly

```
age = 30
length = 1.64
birth_year = 2020 - age
```

Operations with numbers

- addition/subtraction: +, -
- multiplication/division: *, /
- exponent: **
- integer division: //

Relational operators

- greater/less: <,<=,>=,>
- equal: ==
- not equal: !=

For example, to calculate the seconds of a leap year

```
print((365 + 1) * 24 * 60 * 60)
31622400

days = 365
hours = 24
minutes = 60
seconds = 60
leap_year = (days+ 1) * hours * minutes * seconds
print(leap_year)
31622400
```

Operands with strings

As with numbers, some operations can be done

Operations with stings

- adding strings:
  “he” + “llo” → “hello”
- multiplication with strings:
  3* “bye” → “byebyebye”

Relational operands
(alphabetic order)

- greater/less: <,<=,>=,>
- equal: ==
- not equal: !=

Casting between types

For example, to cast a number into a string

```
    age = 30
    age_str = str(age)
```

And to cast it again to int

```
    age_int = int(age_str)
```

Casting types

To change the type of the variable

Different conversions

- into string:
  str(variable)
- into int number:
  int(variable)
- into float number:
  float(variable)

Be careful!

You can not concatenate a number to a string

print(name + age) ERROR!

Let’s do some exercises

First 5 exercises from “Fundamentals exercises”

In short...

- Types can not be mixed when concat
    - print("Hello" + 30) ERROR
- To make more complex programs, we need to think step by step: ALGORITHM

Functions

Function

An independent piece of code that performs a specific task

Until now, we used two functions:

- input()
- print()

Many "synonyms":

- Procedure or subprogram, mainly. Methods are also very similar

Calls to functions

In Python is very easy to call a function.

You must know:

- **Name** of the function
- Required **parameters** (whatever goes between parenthesis)
- What it **returns**

For example, we want to calculate the absolute value of a number:

- **Name**: abs
- **Parameters**: original number
- **Result**: another number (the absolute value of the original number)

Functions’ parameters

Too many or very few parameters:

```
abs()
abs(1,-3)
```

Output:

```
TypeError: abs() takes exactly one
argument (0 given)
TypeError: abs() takes exactly one
argument (2 given)
```

Wrong type of parameter:

```
abs(“a”)
```

Output:

```
TypeError: bad operand type for abs(): 'str'
```

Parenthesis are compulsory!

```
print(abs)
```

Output:

```
<built-in function abs>
```

Creating functions

In Python, functions are **defined** with the reserved word **def**

```
def hello(name):
        return "Hello" + name

message = hello("Amaia")
print(message)
```

Output:

```
Hello Amaia
```

Let’s try it

02-03-function_hello.py

Avoid input() and print() inside

- Parameters for input data
- Return for output data

input() and print() to communicate with the user (in general) in the main program

Return of results (return)

As seen, some functions must return something:

- factorial, sum_numbers

But it is not always compulsory:

- hello

- We return the result of the function with the reserved word **return**.
- Be careful! With the return, the function ends
  - The following code will never be executed
- You can use return in any place
  - if, while, for,..
- But the best place uses to be at the end of the function

Be careful...

- Python is interpreted language
    - It needs to get the function definitions before it’s call

Let’s try it

02-06-functions_return.py

Let’s do some more exercises

Repite the first 5 exercises from “Fundamentals exercises”, now using **functions** (exercise 1.6)

Conditionals

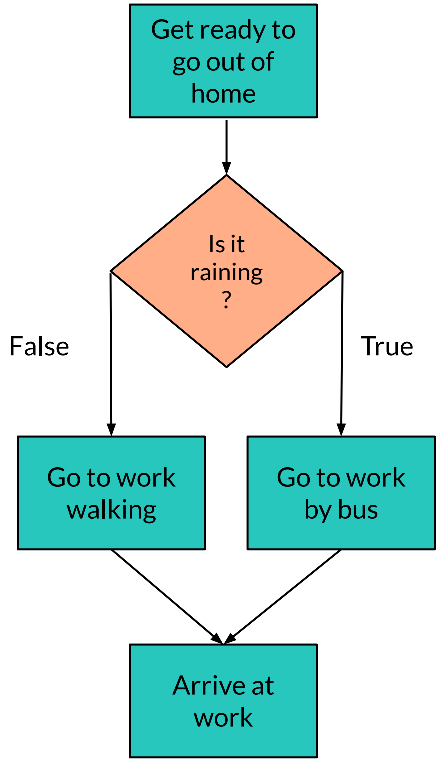
Making decisions

Till now...

We executed operations in order, from to to bottom. Everytime the same will be executed.

But most of the times...

We will want to make decisions depending on something



```
if rain == ‘yes’:
    print(“You better go to work by bus”)
if euria == ‘ez’:
    print(“You better go to work walking”)
```

Let’s try it

02-07-conditionals-rain.py

They are “almost” equivalent...

```
if rain == ‘yes’:
        print(“By bus”)
if rain == ‘no’:
        print(“Walking”)
```

```
if rain == ‘yes’:
        print(“By bus”)
else:
        print(“Walking”)
```

Note

If the two options of the conditional are contraries, you can use the if-else structure.

Let’s try it

02-08-conditionals-rain2.py

Blocks at programming

- When programming, there are blocks that performs a set of operations

Blocks in Python

- The beginning of a block is represented by “two dots” (:)
- The operations inside the block must be indented

```
if rain == ‘yes’:
    print(“It is raining”)
    print(“You better go by bus”)
    print(“You will get wet otherwise”))
else :
    print(“It does not rain”)
    print(“Walking is a good option”)
```

Be careful!

Indentation must be coherent

Usually, 2 or 4 spaces are used

Assignation vs comparison

Assignation

- It is used to assign a value to a variable
- A single = is used
- For example,

```
name = “Olatz”
age = 33
```

Comparison

- It is used to compare two values (it can be the value of variables as well)
- A double = is used
- For example,

```
if name == “Olatz”:
        print(“Kaixo Olatz”)
if name != “Olatz:
        print(“What’s your name?”)
```

Booleans

The result of a comparison will be a boolean value:
True or False

Let’s try it

02-09-booleans.py

Comparisons with numbers

As done with strings, we can write conditionals with numbers

In addition to compare equal (==) and not equal (!=), it is very common to compare greater and less than

- Less than: <
  x < 0
- Less or equal than: <=
  x <= 10
- Greater than: >
  x > 7
- Greater or equal than: >=
  x >= 5

Let’s try it

02-10-booleansNumbers.py

Nested conditionals

A block of ifs can have another block inside

```
if x > 0:
    if x < 10:
        print(“It is a number of one digit”)
    else:
        print(“It has more than one digit”)
else:
    print(“It is a negative number”)
```

Let’s try it

02-11-nestedConditionals.py

Conditionals with many options

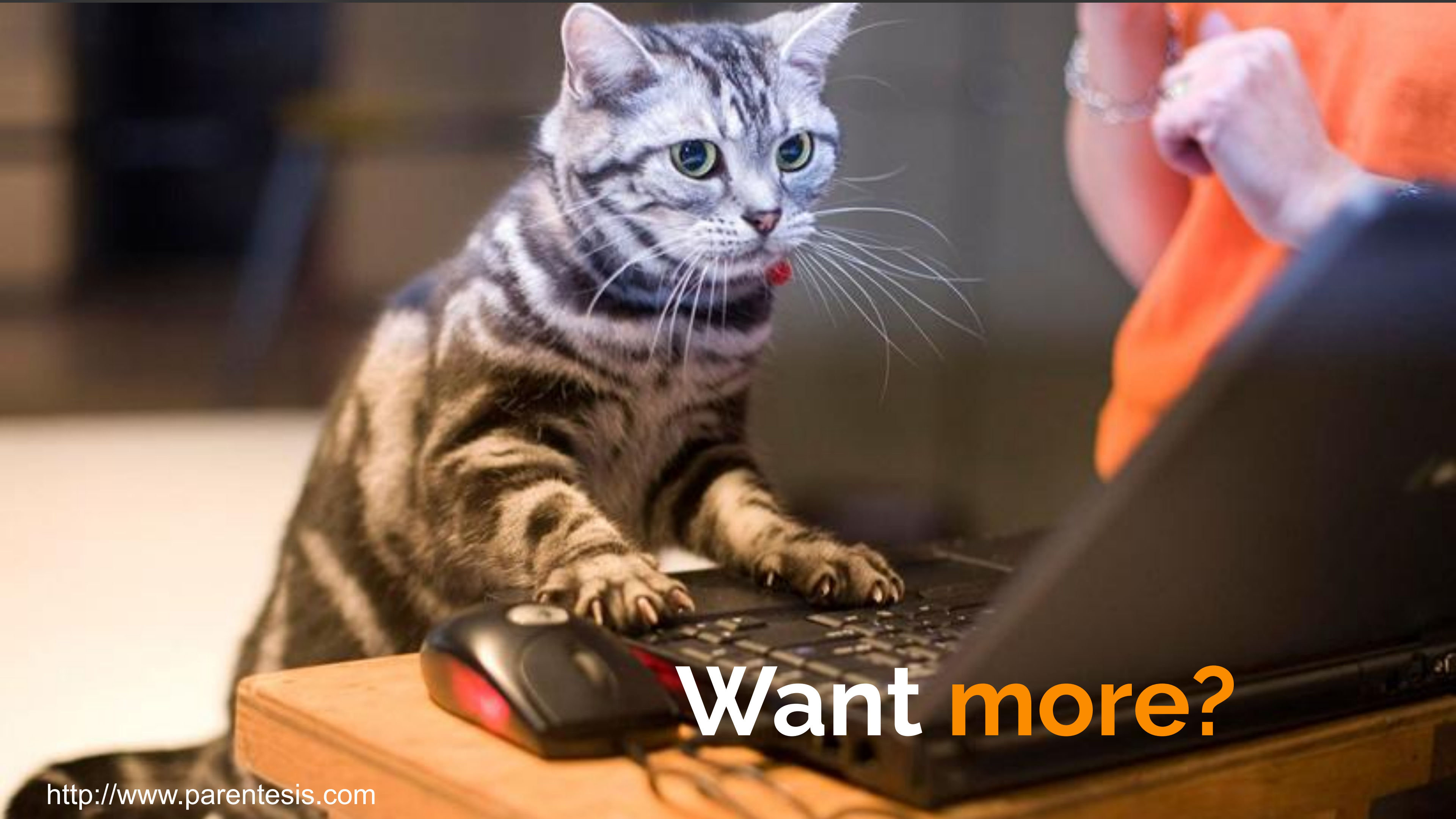
You can concatenate many conditions using the elif structure.
The following two pieces of code are identical.

```
if x > 0:
    print("Positive")
else:
    if x < 0:
        print("Negative")
    else:
        print("Zero")
```

```
if x > 0:
    print("Positive")
elif x < 0:
    print("Negative")
else:
    print("Zero")
```

Let’s try it

02-12-manyConditionals.py



Want more?

http://www.parentesis.com