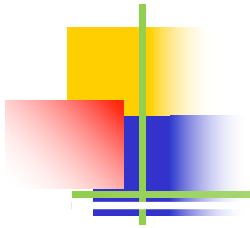

deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Fundamentos de Programação

António J. R. Neves

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

an@ua.pt
http://elearning.ua.pt/

Topics

- Getting started with *Python*
- Values and types
- Variables
- Keywords
- Operators, Expressions and statements
- Functions
- Console input/output
- Scripts

Getting started with *Python* (1)

- *Python* is a general purpose programming language well known for its elegant syntax and readable code.
- With *Python* it is possible to do everything from GUI development, Web applications, System administration tasks, Data Analysis, Visualization, etc.
- *Python* is an ***interpreted*** language - an interpreter parses and executes a *Python* program on a line by line basis. This is usually slower than *compiled* languages.
- However, in *Python* you don't need to define basic data structures, no need to define small utility functions because Python has everything to get you started.
- Moreover *Python* has hundreds of libraries available at `https://pypi.Python.org/`

Getting started with *Python* (2)

- There are two ways to use the interpreter: *interactive mode* and *script mode*. In **interactive mode**, you type *Python* instructions and the interpreter displays the result:

```
>>> 1 + 1
2
```

- The chevron, >>>, is the ***prompt*** the interpreter uses to indicate that it is ready. If you type 1 + 1, the interpreter replies 2.
- Alternatively, you can store code in a file, which is called a ***script***, and use the interpreter to execute it. By convention, *Python* scripts have names that end with `.py`.
- To execute the script, you have to tell the interpreter the name of the file. If you have a script named `test.py` and you are working in a UNIX command window, you type `python3 test.py`. In other development environments, the details of executing scripts are different.

What is a program?

- A program is a sequence of instructions that specifies how to perform a computation. The details look different in different languages, but a few basic types of instruction appear in just about every language:
  - **input**: Get data from the keyboard, a file, or some other device.
  - **output**: Display data on the screen or send data to a file.
  - **math**: Perform basic mathematical operations.
  - **conditional execution**: Check for certain conditions and execute the appropriate code.
  - **repetition**: Perform some action repeatedly, usually with some variation.
- Believe it or not, that is pretty much all there is to it. Every program, no matter how complicated, is made up of instructions that look pretty much like these.

Debugging

- Programming errors are called **bugs** and the process of tracking them down is called **debugging**.
- Three kinds of errors can occur in a program: syntax errors, runtime errors, and semantic errors.
- **Syntax** refers to the structure of a program and the rules about that structure.
- A **runtime error**, only appears after the program has started running. These errors are also called exceptions because they usually indicate that something exceptional (and bad) has happened.
- If there is a **semantic error** in a program, it will run successfully in the sense that the computer will not generate any error messages, but it will not do the right thing. It will do something else.

Values and types

- A **value** is one of the basic things a program works with, like a letter or a number (`33, 3.14, 'ola', 1+2j`).
- Values belong to different **types** (`int, float, str, complex`).
- It is possible to ask the interpreter about it:

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
>>> type(3+5j)
<class 'complex'>
```

- Remember in *Python* everything is an **object**. Even basic data types (to explore later in this course).

Variables

- A **variable** is a name that refers to a value.
- An **assignment statement** creates a new variable and gives it a value.

```
>>> n = 17
>>> pi = 3.1415
```

- The type of a variable is the type of the value it refers to.

```
>>> type(pi)
<class 'float'>
```

- Variable names may include both letters and digits, but they must begin with a letter.
- If you give a variable an illegal name, you get a syntax error:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
```

Keywords

- The interpreter uses **keywords** to recognize the structure of the program, and they cannot be used as variable names.
- *Python2* has 31 keywords (In *Python3*, `exec` is no longer a keyword, but `nonlocal` is).

```
and       else      raise
del       if        continue
from      pass      finally
not       yield     is
while     break     return
as        except    def
elif      import    for
global    print     lambda
or        class     try
with      exec
assert    in
```

Operators, Expressions and statements

- **Operators** are special symbols that represent computations (`+`, `-`, `*`, `/`, `**`, `%`, `<=`, `or`).
- The values combined by operators are called **operands**.
- For a given operator, operands must have compatible types. The result type depends on the operand types.
- An **expression** is a combination of values, variables, and operators.
- A **statement** is a unit of code that the Python interpreter can execute.
- The important difference is that an expression has a value (even if `None`); a statement does not.
- In script mode, an expression, all by itself, has no visible effect (unlike the interactive mode).

Operators (2)

- When more than one operator appears in an expression, the *order of evaluation* depends on the rules of precedence (**PEMDAS**).
- Use parentheses to make it obvious!
- Augmented Assignment Operator `(+=, -=, *=, ...)`
- The `+` operator performs concatenation in **strings**.
- The `*` operator also works on **strings**; it performs repetition. For example, `'Ah'*3` is `'AhAhAh'`.
- It is a good idea to add notes to a program to explain in natural language what the program is doing. These notes are called **comments**, and they start with the **#** symbol.

Expressions (2)

- *Python* allow *simultaneous assignment* syntax like this:
- `name, age, height = "Maria", 21, 1.63`
- *Python* has several *built-in data types*, including
  - Numeric types: `int`, `float`, `complex`
  - Sequences: Strings (`str`), `list`, `tuple`
  - Mappings: Dictionary (`dict`).
- Boolean - In *Python* `True` and `False` are boolean literals. But the following values are also considered
- as false.
  - 0 - zero , 0.0
  - `[]` - empty list , `()` - empty tuple , `{}` - empty dictionary

Functions (1)

- In the context of programming, a **function** is a named sequence of statements that performs a computation.
- In a function definition, the name and the sequence of statements are specified. (We'll get back to this.)
- Later, it can be **called** (or **invoked**) by name.

```
>>> type(32)
<class 'int'>
```

- The name of the function is `type`. The expression in parentheses is called the argument of the function.
- A function “takes” an argument and “returns” a result.

Functions (2)

- *Python* provides built-in functions that convert values from one type to another.
- The `int` function takes any value and converts it to an integer, if it can.
- `float` converts integers and strings to floating-point numbers.
- `str` converts its argument to a string:

```
>>> int('32')
32
>>> int(3.99999)
3
>>> int(-2.3)
-2
>>> float(32)
32.0
>>> float('3.14')
3.14
```

Math functions

- *Python* has a math module that provides most of the familiar mathematical functions.
- A module is a file that contains a collection of related functions.
- Before using a module, it should be imported:

```
>>> import math
>>> print(math)
<module 'math' (built-in)>
```

- To access one of the functions, specify the name of the module and the name of the function, separated by a dot.

```
>>> degrees = 45
>>> radians = degrees / 360.0 * 2 * math.pi
>>> math.sin(radians)
0.707106781187
```

Receiving input from the console

- The `input` function is used to get input from the console.
- It has an optional argument called the *prompt* and returns a string.

```
>>> name = input("What's your name? ")
What's your name? tim
>>> name
'tim'
```

- To get other types of values, you must convert!

```
>> age = int(input(‘Enter your age: ‘))
Enter your age: 22
>>> age
22
>>> type(age)
<class 'int'>
```

Writing to the console

- To output text to the screen, use the `print` function:
```
print("Hello World")
```
- To write multiple lines, add the '\n' character:
```
print("Hello World\nThis is a message")
```
- To print multiple values (separated by blanks):
```
print("speed =", v)
```
- The print function has some optional keyword arguments:
```
print(..., sep=' ',end='\n',file=sys.stdout,flush=False)
```
- The keyword parameter "end“ is used for ending the output of the values.
- Redefining the keyword parameter "file" the output is sent into a different stream (eg. file).
```
>>> fh = open("data.txt","w")
>>> print(“Some text", file=fh)
>>> fh.close()
```

Script Mode Programming

- Invoking the interpreter with a script parameter begins the execution of the script. *Python* files have extension `.py`
- Lines and Indentation - Blocks of code are denoted by line **indentation**, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.
- Statements in *Python* typically end with a new line. *Python* does, however, allow the use of the line continuation character (\) to denote that the line should continue.
- The semicolon (**;**) allows multiple statements on the single line.
- A line containing only whitespace is known as a blank line and *Python* totally ignores it.