A Practical Introduction to Python (and a bit of Sage)

Sebastiano Tronto

2021-04-02

UNIVERSITÉ DU LUXEMBOURG

Second part - schedule

Date	Topics	Homework	Deadline
April 2	Python "review", a bit of Sage	(see March 26)	April 18
April 23	Sage: Algebra and Crypto	Homework 3	May 9
May 7	Sage: Analysis and Statistics	Homework 4	May 30
May 21	Fast code, other software		

UNIVERSITÉ DU LUXEMBOURG

What is programming?

- Software (apps): commands written in a *programming language*
- Programming language: mix of English and symbols
- The code is *compiled* to machine language (C, C++) or *interpreted* by some software in the middle (Python)

UNIVERSITÉ DU LUXEMBOURG

Python

`https://www.python.org`

- Interpreted: slower than C, usable interactively
- Simple syntax, easy to learn
- Very popular
- Sage is based on Python

The interpreter - Python as a calculator

```
~ $ python
Python 3.9.2 (default, Mar  5 2021, 11:40:33)
[GCC 10.2.1 20201203] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> 2+2
4
>>> thanks!
  File "<stdin>", line 1
    thanks!
          ^
SyntaxError: invalid syntax
>>> quit()
~ $
```

The interpreter - Python as a calculator

`https://www.python.org/shell`

- Each line executed as you enter it, result is printed
- Usual math operations work: try them!

```
a+b, a-b, a*b, a/b
a**b (power), a//b (integer division), a%b (remainder)
```

- More math functions:

```
import math
math.sqrt(3)
```

UNIVERSITÉ DU LUXEMBOURG

Help!

- Type `help()` for interactive help
- Try `help("math")` and `help("import")`
- What does

```
from math import *
```

do?

Variables

```
variable_name = value      # This is an assignment
```

- Save results, use them later
- `variable_name`: combination of letters, numbers, underscores
- = always means *assignment*, never *equality*

UNIVERSITÉ DU LUXEMBOURG

Types

```
type(variable_name)      # Get the type of a variable
```

- In other languages you must specify the type of a variable
- Python figures out automatically (*dynamic typing*)
- Each type allows different operations

Other types: Boolean and String

```
my_bool = True
s1 = "hello!"      # Same as 'hello!'
```

- `bool`: `True` or `False`
  - Operations on `bool`: `and`, `or`, `not`
  - Operations with boolean result: `==`, `!=`, `>`, `<`, `>=`, `<=`
- `str`: a string of characters
  - Useful operations:

```
len(str)    # Length, integer value
str + str   # Concatenation
int * str   # Repetition
```

UNIVERSITÉ DU LUXEMBOURG

Lists and sets

```
[2.5, True, "hello"]     # List
{2.5, True, "hello"}     # Set
```

- Lists: keep order and duplicates
- Sets: disregard order and duplicates, allow set operations

UNIVERSITÉ DU LUXEMBOURG

Lists and sets

Things in common (*A is a list or a set*):

- `len(A)`: number of elements (**int**)
- `x in A`: check if `x` is in `A` (**bool**)
- If `A` contains numbers: `max(A)`, `min(A)`, `sum(A)`

Pass from one type to the other: `set(A)` and `list(A)`

UNIVERSITÉ DU LUXEMBOURG

List (and set) comprehension

```
[x**2 for x in [-1,4,1,0] if x < 3]    # Result:  [1,1,0]
{x**2 for x in [-1,4,1,0] if x < 3}    # Result:  {0,1}
```

- Mathematical way to define lists and sets
- Complete syntax:

  `[f(x,y,...)  for x in L for y in M ...  if cond(x,y,...)]`

  - Use as many variables `x, y, ...` as you want
  - `L, M, ...` are lists or sets or other collections
  - `f(x,y,...)` is any expression depending on the variables
  - `cond(x,y,...)` has Boolean value

Lists: access elements and sublists

```
A[i]           # i-th element of A (i ∈ {0,...,len(A) − 1})
A[i] = value   # Change i-th element of A
A[i:j:k]       # Sublist from A[i] to A[j] with step k
A[i:j]         # Same as A[i:j:1]
A[i:]          # Same as A[i:len(A):1]
A[:j]          # Same as A[0:j:1]
```

UNIVERSITÉ DU LUXEMBOURG

List operations

```
A.append(x)     # Append x to A (change A)
A.insert(i,x)   # Insert x in position i (change A)
del A[i]        # Remove i-th element of A (change A)
```

```
A+B  # Concatenation of A and B (list)
A*n  # Repetition of A (list)
```

UNIVERSITÉ DU LUXEMBOURG

Set operations

```
A.add(x)      # Add x to A (change A)
A.remove(x)   # Remove x from A (change A)

A < B (or A <= B)  # A contained in (or equal to) B (bool)
A > B (or A >= B)  # A containes (or is equal to) B (bool)

A | B  # Union (set)
A & B  # Intersection (set)
A - B  # Set difference (set)
```

UNIVERSITÉ DU LUXEMBOURG

Writing more complex programs

test.py

```
1  x = input("Type your answer here: ")
2
3  print("Your answer:", x)
4  print("Counting down...")
5
6  for i in range(int(x),-1,-1):
7      print(i, end=" ")
8
9  print("Done!")
10
```

```
Type your answer here: 42
Your answer: 42
Counting down...
42 41 40 39 38 37 36 35 34 33 32 31
30 29 28 27 26 25 24 23 22 21 20 19
18 17 16 15 14 13 12 11 10 9 8 7 6 5
 4 3 2 1 0 Done!
```

UNIVERSITÉ DU LUXEMBOURG

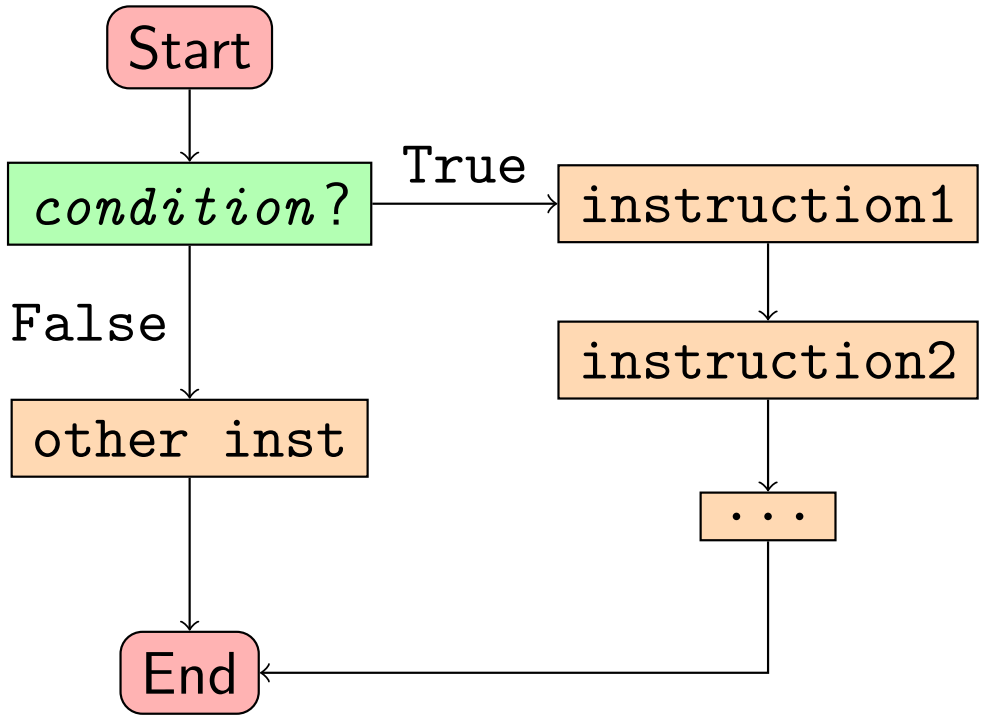
Non-interactive Python

- You can write a file (for example with `https://www.geany.org`)
- Output results with `print("string", or, other, values)`
- Get input (`str`) with `x = input("Prompt:  ")`, convert with `int(x)` or `float(x)`...
- Blocks of code: use *indentation* (see next slides)

UNIVERSITÉ DU LUXEMBOURG

if statement

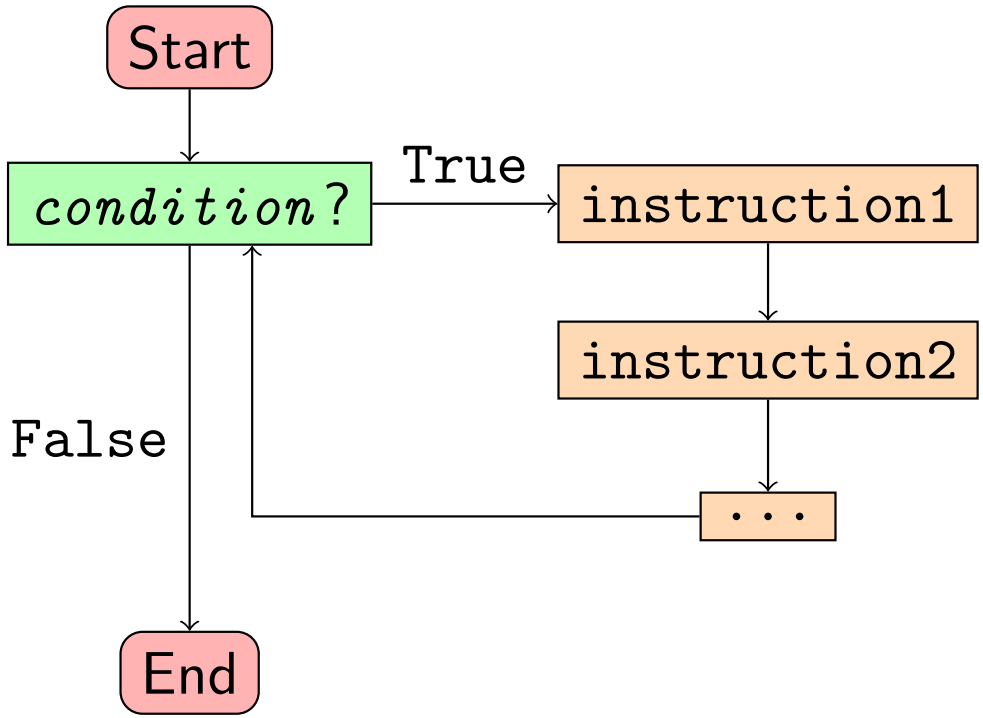
```
if condition:
    instruction1
    instruction2
    ...
else:  # This is optional
    other inst
```



UNIVERSITÉ DU LUXEMBOURG

while loop

```
while condition:
    instruction1
    instruction2
    ...
```



UNIVERSITÉ DU LUXEMBOURG

for loop

```
for i in A:
    instruction1
    instruction2
    ...
```

- Repeats instructions as `i` varies in `A`
- `A` can be list, set or other collection
- Example: `A` can be `range(a,b,step)`

UNIVERSITÉ DU LUXEMBOURG

Functions

```
def f(x, y, ...):
    instruction1
    instruction2
    ...
    return some_value
```

- Useful to divide programs into "pieces"
- The result of `f(x,y,...)` is given by `return ...`

UNIVERSITÉ DU LUXEMBOURG

An example of function (with recursion)

```
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

- Elegant, but slow (in this case)
- Can get stuck in infinite loop: when?

UNIVERSITÉ DU LUXEMBOURG

Sage

`https://www.sagemath.org`

- Mathematical software, uses Python as a language
- Use it interactively or with Jupyter notebook
- Try it online: `https://sagecell.sagemath.org` or `https://cocalc.com/app`

UNIVERSITÉ DU LUXEMBOURG

Differences with Python

Python

```
>>> type(5)
<class 'int'>
>>> 5/2
2.5
>>> type(5/2)
<class 'float'>
>>> type(2.5)
<class 'float'>
>>> 5**3
125
```

Sage

```
sage: type(5)
<class 'sage.rings.integer.Integer'>
sage: 5/2
5/2
sage: type(5/2)
<class 'sage.rings.rational.Rational'>
sage: type(2.5)
<class 'sage.rings.real_mpfr.RealLiteral'>
sage: 5^3
125
```

Sage Documentation

- Tutorial (guided examples): type `tutorial()` or visit `https://doc.sagemath.org/html/en/tutorial`
- `help()`: works as in Python
- Reference manual (detailed technical information): `https://doc.sagemath.org/html/en/reference`

UNIVERSITÉ DU LUXEMBOURG