Introducing Python

Alexis Dimitriadis

1. Over het practicum

- Dit practicum richt zich op het zelfstandig programmeren voor NLP, met Python en in het raamwerk van de Natural Language Toolkit (NLTK).
- In elk practicum gaan we aan de slag met een lijst van taken. De taken worden niet getoetst, de toetsing is bijna geheel gebaseerd op de teamopdrachten.
- **Het is niet de verwachting (en niet nodig) dat alle opdrachten afgerond worden tijdens het practicum.** De taken zijn geschikt voor studenten met verschillende programmeer ervaring, en er kan ook thuis aan gewerkt worden.
- **Dit practicum is geen complete inleiding tot programmeren.** Het boek *Think Python* biedt een uitgebreide introductie tot de nodige concepten.

  Heb je weinig ervaring met programmeren dan wordt sterk aangeraden om voor elk practicum de betreffende hoofdstukken van *Think Python* goed te lezen en zo mogelijk door te werken.

2. What is python, and why use it?

- A “high level” programming language, known for its readability, ease of use, and powerful libraries.

```
print("Hello world!")
```

- Easy to learn and to read.
- Interactive mode allows experimentation and exploration, from the python prompt or from IPython Notebook.
- Well-suited to short programs (as well as large ones).
- The functionality of the basic language is extended by many “libraries”, including the *Natural Language Toolkit (NLTK)*.
- Is becoming a standard choice for language processing (replacing perl).

- Compare Python to the C# “hello world” program:

```
using System;
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

- Python:

```
print("Hello world!")
```

- Instead of declaring a fixed type for each variable (as in Java, C, etc.), python allows any type to be stored:

```
x = 10
x = "Hello"
x = [ 2, 3, 4, "done" ]
```

- The downside of this flexibility is that python is slower than these stricter languages, since it must keep track of the type of data at run time.

3. Some rules of python

- Variables are normally invisible to us. We must ask python to **print** them.
- Variable names **must start with a letter.** They can include letters, digits and underscores (_). No spaces or punctuation!
- The symbol # begins a **comment:** Everything from the # to the end of the line is intended only for human readers. It is ignored by python, and has no effect on the operation of the program.

```
print(3.0 * 3)      # What will this print?
```

- The decimal point in numbers is written with a dot (American style).
- Text strings can be enclosed in either single or double quotes (with the same meaning): `'string'` or `"string"`

4. “Scalar” data types

Data types that (conceptually) contain just one value.
(NB: Strings count as a scalar data type.)

`int`	integer (whole number)	`-3, 4, 23772, ...`
`float`	real number	`1.0, 3.336, -0.0000004, ...`
`bool`	boolean	`True, False`
`str`	string of characters	`"hello", 'No way', '"Yes!", she said'`
`None`	“nothing”	`None`

5. Integers (int): Arithmetic operators

```
print(2+2)          # sum: prints 4
print(4-15)         # difference: prints -11
print(2*3)          # multiplication: prints 6
```

- division:

```
print(5 / 3)        # division: prints 1.66667
print(5//3)         # integer division: prints 1
print(5.0//3)       # also integer division! prints 1.0
```

- modulo: the remainder of integer division

```
print(14 % 3)       # prints 2, since 14 == 3*4 + 2
```

- exponentiation: "to the *n*th power"

```
print(2**4)         # prints 16
```

6. Real numbers (float)

```
print(3.14159)
print(2.0 + 5)    # prints 7.0
print(2.0 / 3)    # prints 0.666666667
print(1.35e6)     # prints 1350000.0
print(float(4))   # prints 4.0
```

7. Booleans (bool)

- Expressions that express whether or not something is true (`True`/`False`).

```
print(not True)          # prints False
print(4 > 3)             # prints True

print(2 + 2 == 4)        # prints True
print(3 + 2 != 32)       # prints True

print((-5 == 5-10) and (5 <= 10))      # prints True
print((-5 == 5-10) and not (5 < 10))   # prints False
```

- Boolean values are ordinary values, and can be assigned to variables:

```
result = (1 < 2 or 2+2 == 5)

print(result)            # prints True
```

8. if statement: check status

```
1 if 33*44 > 34*43:
2     print("The first expression is bigger")
3     print("Their difference is", 33*44 - 34*43)
4 elif 33*44 == 34*43:
5     print("They are equal!")
6 else:
7     print("The second expression is bigger")
```

- **NOTE the indentation!** Python uses indentation to group statements, not braces `{, }`. People rely on indentation for visual grouping, and mismatches between braces and indentation are a frequent source of errors. It is better to have only one system.

- The condition is in principle a boolean expression (true or false), but any variable or expression can be used.

9. What is “truth”?

- A number counts as “true” if its value is not 0 (or 0.0).

```
messages = 3
if messages:
    print("You have mail")
```

- A string counts as “true” if it is not the empty string (`""`, length 0).

- `None` and (of course) `False` count as false; almost all other values count as true.

```
student_id = None
name = ""
if name:
    print("Hello," name)
elif student_id:
    print("Hello," student_id)
```

10. Comparison operators

- Comparison operators are used in logical expressions. Note that equality is written "==", since "=" is used for assignment.

```
x == y      # x is equal to y (x = y)
x != y      # x ≠ y
x > y
x < y
x >= y      # x ≥ y
x <= y      # x ≤ y
```

- Used for numbers, but also for strings; in that case, comparison is based on alphabetical order.

- Also handy, for checking (sub)strings:

```
'bon' in 'Lisbon'
'i' not in 'team'
```

Contents