Hello Python!

Dr. Giuseppe Maggiore

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Lecture topics

- We introduce Python
- We bridge what we have seen in the previous lecture with actual Python elements

Why many programming languages?

- Low-level vs high-level
- Statically-typed vs dynamically-typed
- Compiled vs interpreted
- Imperative vs functional vs logic vs declarative vs object-oriented
- Safe vs unsafe
- Fast vs slow
- ...

Introduction

Why many programming languages?

- The set of all problems is a complex, fractal-looking shape
- The programming language we choose shifts our focus on these problems
- Some become more visible and obvious to solve...

Introduction

Why many programming languages?

- The set of all problems is a complex, fractal-looking shape
- The programming language we choose shifts our focus on these problems
- Some become more visible and obvious to solve...
- ...others become hidden, obstructed, or harder to solve

Introduction

Why many programming languages?

- Not all languages are equal
- There is improvement and an ordering
    - For low-level programming C is in most cases better than assembly
    - For data transformation SQL is in most cases better than Java
    - For algorithmic work on trees F# is in most cases better than C#

Why many programming languages?

- Not all languages are comparable
- There are perfectly valid differences in balance and features
  - Most languages are better than assembly in most scenarios
  - For data transformation SQL is as good as F# on algorithmic work on trees

Introduction

Early programming languages

- Analytical Engine/Difference Engine: hypothetical mechanical computers (1840's, Charles Babbage and Ada Lovelace)
- Assembly language: programming close to the machine (1940's)
- Fortran, ALGOL, and COBOL: various forms of imperative programming (1950's)
- LISP: functional and meta-programming (1950's, still in use)
- Simula: object-oriented programming (1950's)
- C: high-level low-level programming (1970's, still in use)
- Smalltalk: everything-is-an-object programming (1970's)
- Prolog: logic programming (1970's)

Introduction

1980's

- C++: C with classes (still in use)
- Matlab and Mathematica: mathematics and simulations (still in use)
- Erlang: concurrency and telecommunications (still in use)

1990's: the Internet Age

- Haskell: functional programming (still in use)
- Python, Ruby, Lua: concise, dynamic programming (still in use)
- JavaScript: webpage dynamics (still in use)
- Java: objects and portability (still in use)

2000's: the Modern Age

- C#: objects and portability (still in use)
- F# and Scala: hybrid, functional-first programming and portability (still in use)
- Go and Swift: native, safe development (getting traction?)

The Python Zen

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Python introduction

- General-purpose language
- High-level
- Concise on purpose
- Dynamically typed
- Hybrid paradigm, imperative/procedural first

The Python Programming Language

Why Python?

- Used a lot as a beginning languages in higher education
- Adequate for expressing the basics of computational thinking
- High signal to noise ratio of syntax

Python basic syntax and semantics

Variables

- Variables are not declared
- Just initialize and subsequently use

Variable names

- Variables may begin with any letter or the _ sign
- Followed by any sequence of letters, numbers, and the _

```
x
y
_x
customer_name
_x1
_x1_customer
```

Variable names

- Python supports integers and other sorts of numbers
- Any sequence of numeric characters (we call it an integer *literal*) is a number

```
100
0
-1
79228162514264337593543950336L
```

Variable names

- We can assign a value to a variable
- `variableName = expression`
- What does this do to the memory of the program? **Discuss.**

Variable names

- We can assign a value to a variable
- `variableName = expression`
- What does this do to the memory of the program? **Discuss.**
- **If the variable did not exist, then we add it to memory**
- If the variable existed, then we change its value in memory

$$\begin{cases} (PC, S) \stackrel{x=e}{\rightarrow} (PC+1, S'[x \mapsto e]) & when \quad x \in S \land S' = S - \{x\} \\ (PC, S) \stackrel{x=e}{\rightarrow} (PC+1, S[x \mapsto e]) & when \quad x \in S \end{cases}$$

PC
1

```
x = 100
y = 200
z = 50
```

what changes while running the current instruction? **Try to guess and discuss!**

PC
1

```
x = 100
y = 200
z = 50
```

PC
1

```
x = 100
y = 200
z = 50
```

PC	x
2	100

PC	x
2	100

```
x = 100
y = 200
z = 50
```

PC	x
2	100

```
x = 100
y = 200
z = 50
```

PC	x	y
3	100	200

PC	x	y
3	100	200

```
x = 100
y = 200
z = 50
```

PC	x	y
3	100	200

```
x = 100
y = 200
z = 50
```

PC	x	y	z
4	100	200	50

Variable names

- We can assign a value to a variable
- `variableName = expression`
- What does this do to the memory of the program? **Discuss.**

Variable names

- We can assign a value to a variable
- `variableName = expression`
- What does this do to the memory of the program? **Discuss.**
- If the variable did not exist, then we add it to memory
- **If the variable existed, then we change its value in memory**

$$\begin{cases} (PC, S) \stackrel{x=e}{\rightarrow} (PC+1, S'[x \mapsto e]) & when \quad x \in S \wedge S' = S - \{x\} \\ (PC, S) \stackrel{x=e}{\rightarrow} (PC+1, S[x \mapsto e]) & when \quad x \in S \end{cases}$$

PC	x	y	z
1	0	-1	5

```
x = 100
y = 200
z = 50
```

what changes while running the current instruction? **Try to guess and discuss!**

PC	x	y	z
1	0	-1	5

```
x = 100
y = 200
z = 50
```

PC	x	y	z
1	0	-1	5

```
x = 100
y = 200
z = 50
```

PC	x	y	z
2	100	-1	5

PC	x	y	z
2	100	-1	5

```
x = 100
y = 200
z = 50
```

PC	x	y	z
2	100	-1	5

```
x = 100
y = 200
z = 50
```

PC	x	y	z
3	100	200	5

PC	x	y	z
3	100	200	5

```
x = 100
y = 200
z = 50
```

PC	x	y	z
3	100	200	5

```
x = 100
y = 200
z = 50
```

PC	x	y	z
4	100	200	5 0

This is it!

The best of luck, and thanks for the attention!