Introduction to Programming using PYTHON

Session 1

N. D. Mendes
ndm@algos.inesc-id.pt

INESC-ID

October 4, 2006

Part I

About the course

About the course

- The course will consist of 10 sessions including this one

About the course

- The course will consist of 10 sessions including this one
- The course is **very** introductory

About the course

- The course will consist of 10 sessions including this one
- The course is **very** introductory
- You will be asked to do much of the work outside the course

About the course

- The course will consist of 10 sessions including this one
- The course is **very** introductory
- You will be asked to do much of the work outside the course
- We will be following a manual prepared at Institut Pasteur
  - http://www.pasteur.fr/formation/infobio/python/

About the course

- The course will consist of 10 sessions including this one
- The course is **very** introductory
- You will be asked to do much of the work outside the course
- We will be following a manual prepared at Institut Pasteur
  - http://www.pasteur.fr/formation/infobio/python/
- Before every session you will be asked to:
  - read a number of chapters from the manual
  - complete a number of exercises

Part II

Basic Programming Rules

Basic Programming Rules

- Programming is unnatural

Basic Programming Rules

- Programming is unnatural
  - Requires abstract reasoning
  - Requires the use of an unambiguous language

Basic Programming Rules

- Programming is unnatural
  - Requires abstract reasoning
  - Requires the use of an unambiguous language
- Programs should be **elegant** and **readable**

Basic Programming Rules

- Programming is unnatural
  - Requires abstract reasoning
  - Requires the use of an unambiguous language
- Programs should be **elegant** and **readable**
  - Elegant programs

    Programs that address the problem at hand with simplicity, being concerned with the general case and addressing the exceptions separately. Elegant programs do not try to do everything in a single step and tend to be modular.
  - Readable programs

    Programs that can be easily followed and readily understood by someone who knows the language. The code is simple. Programs with commentaries tend to be readable.

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy
   - Do not repeat code unnecessarily
   - Make extensive use of helper procedures

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy
   - Do not repeat code unnecessarily
   - Make extensive use of helper procedures
3. Give meaningful names to variables, functions, etc

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy
   - Do not repeat code unnecessarily
   - Make extensive use of helper procedures
3. Give meaningful names to variables, functions, etc
   - `temperature` and `size` are much better than `x` or `y`
   - Exceptions can be made for iteration variables (e.g. `i`) which are generally intrinsically meaningless

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy
   - Do not repeat code unnecessarily
   - Make extensive use of helper procedures
3. Give meaningful names to variables, functions, etc
   - `temperature` and `size` are much better than `x` or `y`
   - Exceptions can be made for iteration variables (e.g. `i`) which are generally intrinsically meaningless
4. Print meaningful error messages

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy
   - Do not repeat code unnecessarily
   - Make extensive use of helper procedures
3. Give meaningful names to variables, functions, etc
   - `temperature` and `size` are much better than `x` or `y`
   - Exceptions can be made for iteration variables (e.g. `i`) which are generally intrinsically meaningless
4. Print meaningful error messages
   - Messages like `"An error has occurred..."` are as good as nothing

Basic Programming Rules

Rules of Thumb

1. Do not re-invent the wheel
   - Other people have already done it
   - Chances are that they have done a better job than you would
2. Be lazy
   - Do not repeat code unnecessarily
   - Make extensive use of helper procedures
3. Give meaningful names to variables, functions, etc
   - `temperature` and `size` are much better than `x` or `y`
   - Exceptions can be made for iteration variables (e.g. `i`) which are generally intrinsically meaningless
4. Print meaningful error messages
   - Messages like `"An error has occurred..."` are as good as nothing
5. Document your programs and functions

Part III

Structured Programming

Structured Programming

The notion of Structured Programming is not universal. In Dijkstra's sense it can be summarized into these principles:

Structured Programming

The notion of Structured Programming is not universal. In Dijkstra's sense it can be summarized into these principles:

- There should only be 3 ways of combining program instructions: **sequencing**, **selection** and **iteration**

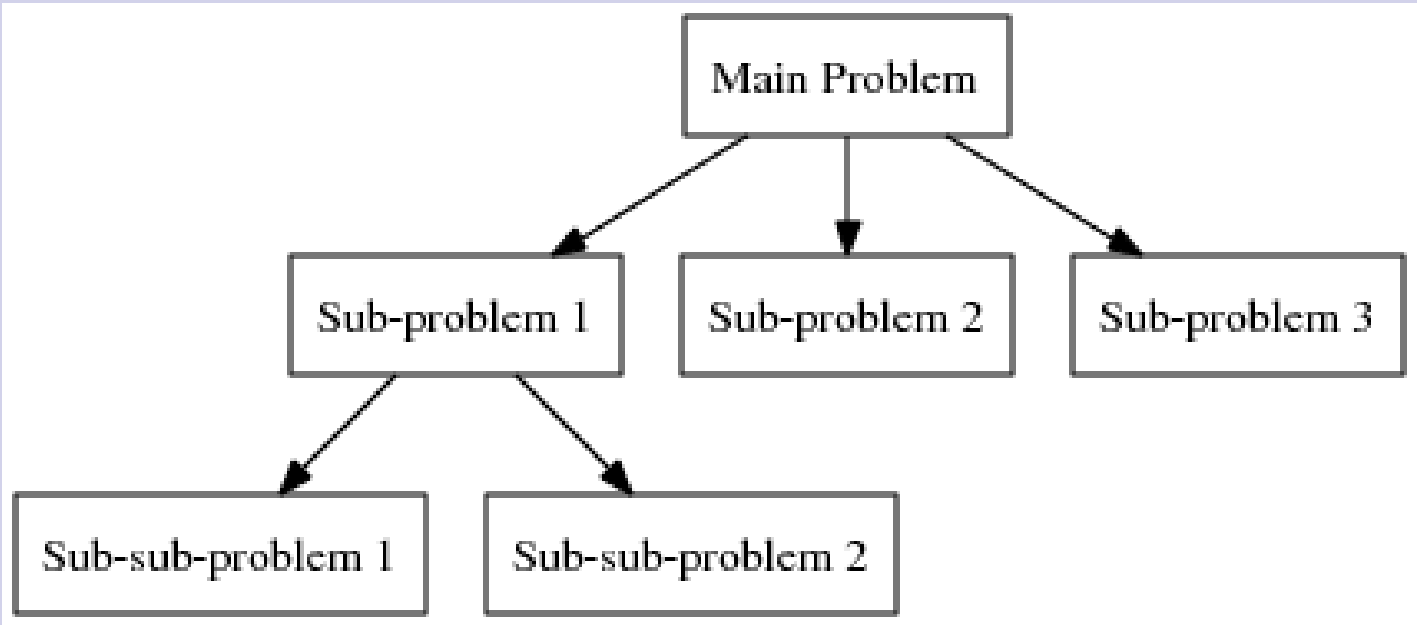
Sequencing	Selection	Iteration

instruction$_1$
instruction$_2$
...
instruction$_n$
	if $condition$ **then**
	instruction$_{t_1}$
	instruction$_{t_2}$
	...
	instruction$_{t_n}$
	else
	instruction$_{f_1}$
	instruction$_{f_2}$
	...
	instruction$_{f_n}$
	end if
		while $condition$ **do**
		instruction$_1$
		instruction$_2$
		...
		instruction$_n$
		end while

Structured Programming

The notion of Structured Programming is not universal. In Dijkstra's sense it can be summarized into these principles:

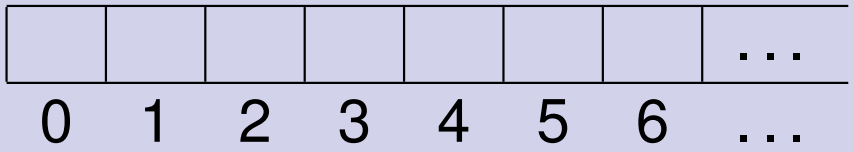
- There should only be 3 ways of combining program instructions: **sequencing**, **selection** and **iteration**
- We should adopt a top-down approach whereby we divide a problem into smaller subproblems which can eventually be directly addressed
  - Programs should be broken down into modules, functions and blocks which are easy to understand



Part IV

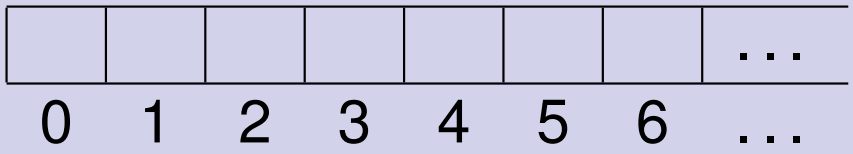
A taste of abstract thinking

URM



- Abstract machine (Unlimited Register Machine)
- Each register R_i contains an integer
- The machine keeps an additional counter p for the index of the next instruction
- A URM program consists of an ordered list of commands:

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p + 1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p + 1$
$T(j, i)$	$R_j \leftarrow R_i$	$p \leftarrow p + 1$
$J(i, j, n)$	no effect	$p \leftarrow n$ if $R_i = R_j$
		$p \leftarrow p + 1$ if $R_i \neq R_j$



Conventions

- Initially $R_i \leftarrow 0, \forall_i$
- The output of a program is given by R_0
- The the n arguments of a program are given at $R_1, \ldots, R_n$
- The working space of a program are all registers $R_i, i > n$

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p + 1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p + 1$
$T(j, i)$	$R_j \leftarrow R_i$	$p \leftarrow p + 1$
$J(i, j, n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p + 1$ if $R_i \neq R_j$

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO
- Which command can be implement at the expense of the others?

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO
- Which command can be implement at the expense of the others?
  $T(j,i)$

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO
- Which command can be implement at the expense of the others?
  $T(j,i)$
- How?

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$
		$p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO
- Which command can be implement at the expense of the others?
  $T(j,i)$
- How?

1: $J(i,j,6)$
2: $Z(j)$
3: $J(i,j,6)$
4: $S(j)$
5: $J(0,0,3)$
6: HALT

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$
		$p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO
- Which command can be implement at the expense of the others?
  $T(j,i)$
- How?

1: $J(i,j,6)$
2: $Z(j)$
3: $J(i,j,6)$
4: $S(j)$
5: $J(0,0,3)$
6: HALT

- Why do we need the first command?

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$
		$p \leftarrow p+1$ if $R_i \neq R_j$

- Are all these commands really necessary?
  NO
- Which command can be implement at the expense of the others?
  $T(j,i)$
- How?

1: $J(i,j,6)$
2: $Z(j)$
3: $J(i,j,6)$
4: $S(j)$
5: $J(0,0,3)$
6: HALT

- Why do we need the first command?
  Consider the case where $i = j$

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

How would you implement **addition** such that in the end of the program $R_0 \leftarrow R_1 + R_2$

Command	Effect on registers	Effect on program counter
$Z(i)$	$R_i \leftarrow 0$	$p \leftarrow p+1$
$S(i)$	$R_i \leftarrow R_i + 1$	$p \leftarrow p+1$
$T(j,i)$	$R_j \leftarrow R_i$	$p \leftarrow p+1$
$J(i,j,n)$	no effect	$p \leftarrow n$ if $R_i = R_j$ $p \leftarrow p+1$ if $R_i \neq R_j$

How would you implement **addition** such that in the end of the program $R_0 \leftarrow R_1 + R_2$

1: $T(0,1)$
2: $J(2,3,6)$
3: $S(0)$
4: $S(3)$
5: $J(0,0,2)$
6: HALT

Let us computer $2 + 2$

0	2	2	0	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

2	2	2	0	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0,1)$
2: $J(2,3,6)$
3: $S(0)$
4: $S(3)$
5: $J(0,0,2)$
6: HALT

Let us computer $2 + 2$

2	2	2	0	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

3	2	2	0	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

3	2	2	1	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

3	2	2	1	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

3	2	2	1	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

4	2	2	1	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

4	2	2	2	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

4	2	2	2	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

4	2	2	2	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Let us computer $2 + 2$

$\underline{4}$	2	2	2	0	0	0	...
0	1	2	3	4	5	6	...

1: $T(0, 1)$
2: $J(2, 3, 6)$
3: $S(0)$
4: $S(3)$
5: $J(0, 0, 2)$
6: HALT

Part V

For the next session

- From the manual
  - Chapters 1, 2, 3
- Exercises
  - Implement
    - Multiplication ($R_0 \leftarrow R_1 \times R_2$)
    - $\leq$-test ($R_0 \leftarrow 1$ if $R_1 \leq R_2$ otherwise $R_0 \leftarrow 0$)
    - Subtraction $R_0 \leftarrow R_1 - R_2$
  - Send implementations in **plain text** to *ndm@algos.inesc-id.pt*