



Une école
en mouvement

IN101: Algorithmique et Programmation

F. Brandner, A. Gepperth, T. Hecht, F. Stulp, V. Paun
ENSTA ParisTech

École Nationale Supérieure
de **Techniques Avancées**

License CC BY-NC-SA 2.0



<http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>



What is an algorithm?

- An algorithm is like a cooking recipe

La galette des rois

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferr

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et cacher la ferr!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 180°.

What is an algorithm?

- An algorithm is like a cooking recipe
 1 Input (80gr de sucre, etc.)

La galette des rois

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferr

Input

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et cacher la fève!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 180°.

What is an algorithm?

La galette des rois

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferr

Input

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et couvrir la faire!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 170°C

Instructions

- An algorithm is like a cooking recipe
 (80gr de sucre, etc.)

1

Input

2

Instructions

What is an algorithm?

La galette des rois

Output

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferr

Input

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et cacher la fève!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 170°C

Instructions

- An algorithm is like a cooking recipe
 - 1 Input (80gr de sucre, etc.)
 - 2 Instructions
 - 3 Output (galette des rois)

What is an algorithm?

La galette des rois

Output

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferru

Input

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et couvrir la ferru!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 170°C

Instructions

- An algorithm is like a cooking recipe
 - 1 Input (80gr de sucre, etc.)
 - 2 Instructions
 - 3 Output (galette des rois)
- Example from computer science
 - sorting ("tri")

What is an algorithm?

La galette des rois

Output

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferru

Input

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et couvrir la faire!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 170°C

Instructions

- An algorithm is like a cooking recipe
 - ① Input (80gr de sucre, etc.)
 - ② Instructions
 - ③ Output (galette des rois)
- Example from computer science
 - sorting ("tri")

Input:

5 1 4 2 3

What is an algorithm?

La galette des rois

Output

• Ingrédients - 2 rouleaux de pâte feuilletée
 - 1 sachet de poudre d'amande
 - 100 gr de beurre
 - 80 gr de sucre
 - 2 œufs
 - 1 ferru

Input

- 1) Mélanger le sucre, les œufs, la poudre d'amande et le beurre
- 2) Étaler la frangipanne sur l'une des pâtes et couvrir la faire!
- 3) Poser la 2^{ème} pâte dessus et coller les bords. Décorer la galette avec un couteau.
- 4) Faire cuire 30 min à 170°C

Instructions

- An algorithm is like a cooking recipe
 - 1 Input (80gr de sucre, etc.)
 - 2 Instructions
 - 3 Output (galette des rois)
- Example from computer science
 - sorting ("tri")

Input:

5 1 4 2 3



Instructions



Output:

1 2 3 4 5

Example Algorithm: Selection Sort

Instructions

Go through all positions from front to back and do the following:

Find the number with the smallest value starting after the current position

Swap that smallest number with the number in the current position

5	1	4	2	3
---	---	---	---	---

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the smallest value starting after the current position

Swap that smallest number with the number in the current position

5	1	4	2	3
---	---	---	---	---

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that smallest number with the number in the current position



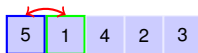
Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position

1	5	4	2	3
---	---	---	---	---

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position



1	5	4	2	3
---	---	---	---	---

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position



1	2	4	5	3
---	---	---	---	---

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position



Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position

1	2	3	4	5
---	---	---	---	---

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position

Swap that **smallest** number with the number in the **current** position

- You call a sorting algorithm dozens of times a day!

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position

- You call a sorting algorithm dozens of times a day!

Sort search results by relevance



Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position

- You call a sorting algorithm dozens of times a day!

Sort hotels by price or stars

597 établissements sur 2463 sont disponibles dans votre destination : Paris.
 Résultats 1 – 15

[Commander](#)
[Comparer](#)
[Index](#)
[Sites d'annuaire](#)
[Trier](#)
[Notes des commentaires](#)
[55 Likes](#)
[Carte](#)



Hotel Paris Arc de Triomphe ★★★★★ [❤️](#) [👍](#)
 Centre de Paris, 18, Champs-Élysées, République, Paris
 Il y a 16 personnes qui regardent la page de cet hôtel
 Dernière réservation: le 17 mai
Etablissement éligible pour notre site

[Supprimer](#) **8,7**
 Note sur 100 commentaires



Hotel du Collectionneur Arc de Triomphe ★★★★★ [❤️](#) [👍](#)
 Centre de Paris, 41, Avenue de la République, République, Paris
 Il y a 6 personnes qui regardent la page de cet hôtel
 Dernière réservation: le 16 mai
 Cherchez Le King Hotel République
 1 type de chambre en stock

[Supprimer](#) **7,8**
 Note sur 49 commentaires



Hotel Marignan Paris ★★★★★ [❤️](#) [👍](#)
 Centre de Paris, 1, Avenue de la République, République, Paris
 Il y a 16 personnes qui regardent la page de cet hôtel
 Dernière réservation: le 17 mai
 Cherchez Le King Hotel République
 1 type de chambre en stock

[Supprimer](#) **8,0**
 Note sur 100 commentaires

Example Algorithm: Selection Sort

Instructions

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position

- You call a sorting algorithm dozens of times a day!
 - It is not magic... somewhere, a sorting algorithm is doing the work

Sort hotels by price or stars

597 établissements sur 2463 sont disponibles dans votre destination : Paris.

Résultats 1 - 15

Classer par: **Recommandé** **Meilleures offres** **Meilleures notes** **Meilleures photos** **Meilleures cartes**

Superior 8.7
Note sur 244 commentaires

Hotel Paris Arc de Triomphe ★★★★★ **Super**
Centre de Paris, 18, Champs-Élysées - Marais, Paris
Il y a 10 personnes qui regardent la page de cet hôtel.
Dates de réservation : le 17 mai
Établissement épuisé sur notre site

Hotel du Collectionneur Arc de Triomphe ★★★★★ **Super**
Centre de Paris, 18, Champs-Élysées - Marais, Paris
Il y a 0 personnes qui regardent la page de cet hôtel.
Dates de réservation : le 18 mai
Chambres et lit très bien équipés
A l'opposé de la chambre au plus

Hotel Marignan Paris ★★★★★ **Super**
Centre de Paris, 18, Champs-Élysées - Marais, Paris - Accès aux métros

Go through **all positions** from front to back and do the following:

Find the number with the **smallest** value starting after the **current** position
Swap that **smallest** number with the number in the **current** position

- You call a sorting algorithm dozens of times a day!
 - It is not magic... somewhere, a sorting algorithm is doing the work
 - Algorithms automate repetitive work, if we **program** them

Sort hotels by price or stars

597 établissements sur 2463 sont disponibles dans votre destination : Paris.
 Résultats 1 – 15

[Commander](#)
[Comparer](#)
[Filtres](#)
[Sites d'avis clients](#)
[Trier](#)
[Notes des commentaires](#)
[55000](#)
[Lien](#)
[Carte](#)



Hotel Paris Arc de Triomphe ★★★★★ [Lien](#)
 Centre de Paris, 18, Champs-Élysées, 06000 Paris
 Il y a 16 personnes qui regardent la page de cet hôtel
 Dernière réservation: le 17 mai
Etablissement éligible pour notre site



Hotel du Collectionneur Arc de Triomphe ★★★★★ [Lien](#)
 Centre de Paris, 41, Avenue de la Grande Armée, 06000 Paris
 Il y a 6 personnes qui regardent la page de cet hôtel
 Dernière réservation: le 16 mai
 Chers Les Très Très Indignes
 1 type de chambre en stock



Hotel Marignan Paris ★★★★★ [Lien](#)
 Supérieur 8,0

What is programming?

- Represent algorithms in a formal language that
 - Humans can (learn to) understand and write
 - Computers can read and execute
- Example languages: C, C++, Java, Matlab, Python

What is programming?

- Represent algorithms in a formal language that
 - Humans can (learn to) understand and write
 - Computers can read and execute
- Example languages: C, C++, Java, Matlab, Python

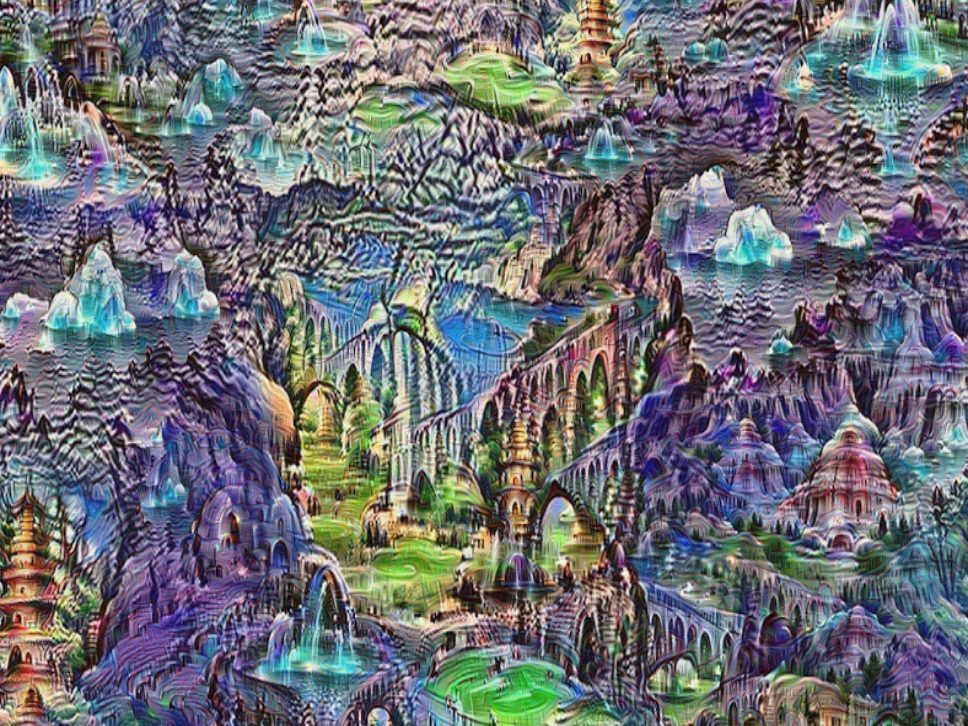
```
# Finding the smallest number  
# in a list of numbers in Python  
  
# Input  
an_input = [8, 9, 6, 5]  
print(an_input)  
  
# Instructions  
smallest = an_input[0]  
for number in an_input:  
    if (number < smallest):  
        smallest = number  
  
# Output  
output = smallest  
print(output)
```

Example Algorithm: Inceptionism

- “Inceptionism: Going Deeper into Neural Networks”

<http://googleresearch.blogspot.co.uk/2015/06/inceptionism-going-deeper-into-neural.html>

- Neural networks that dream!
- So what do they dream? Example...



Outline

- 1 Algorithms
- 2 Programming
- 3 Cours IN101
 - Why IN101?
 - Objectives
 - Modalités
- 4 Python Basics
- 5 TD

IN101 Algorithmique et Programmation: Why?

Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!

- Many programs are called every time you:
 - do an Internet search
 - use your smartphone
 - drive your car



IN101 Algorithmique et Programmation: Why?

Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!

- Many programs are called every time you:
 - do an Internet search
 - use your smartphone
 - drive your car (even more if your car drives by itself!)



IN101 Algorithmique et Programmation: Why?

Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!



Mastère Spécialisé
Capteurs, Géolocalisation
et Navigation



Groupe ENSTA



IN101 Algorithmique et Programmation: Why?

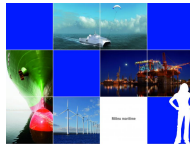
Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!



Mastère Spécialisé
Capteurs, Géolocalisation
et Navigation



Mastère Spécialisé
Génie Maritime :
transport, énergie, développement durable



IN101 Algorithmique et Programmation: Why?

Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!



Mastère Spécialisé
Capteurs, Géolocalisation
et Navigation



Mastère Spécialisé
Génie Maritime :
transport, énergie, développement durable



Master International
Génie Maritime
International Master in
Maritime Engineering



IN101 Algorithmique et Programmation: Why?

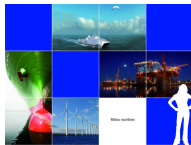
Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!



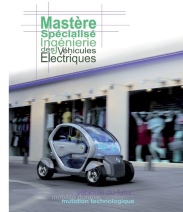
**Mastère Spécialisé
Capteurs, Géolocalisation
et Navigation**



**Mastère Spécialisé
Génie Maritime :**
transport, énergie, développement durable



**Master International
Génie Maritime**
International Master in
Maritime Engineering



IN101 Algorithmique et Programmation: Why?

Automation is everywhere!

⇒ Algorithms are everywhere!

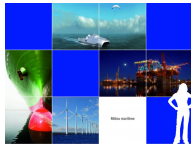
⇒ Programming is everywhere!



**Mastère Spécialisé
Capteurs, Géolocalisation
et Navigation**



Groupe ENSTA



**Mastère Spécialisé
Génie Maritime :**
transport, énergie, développement durable



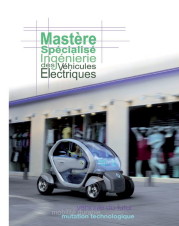
École Nationale Supérieure
de Techniques Avancées



**Master International
Génie Maritime**
International Master in
Maritime Engineering



ENSTA Group



**Master of Science
Water, Air, Pollution
and Energy**



IN101 Algorithmique et Programmation: Why?

Automation is everywhere!

⇒ Algorithms are everywhere!

⇒ Programming is everywhere!

Two robots cooking pancakes, with algorithms written in Python and C++.



IN101 Algorithmique et Programmation: Objectives

- Objectives are to learn to
 - formulate algorithms from problem descriptions
 - implement algorithms in a programming language
 - the programming language being Python
 - apply several common algorithms in computer science
 - formulate and implement algorithms **autonomously**

IN101 Algorithmique et Programmation: Objectives

- Objectives are to learn to
 - formulate algorithms from problem descriptions
 - implement algorithms in a programming language
 - the programming language being Python
 - apply several common algorithms in computer science
 - formulate and implement algorithms **autonomously**
- Objectives are **not** to learn
 - how algorithms/programs are executed on a computer (→ IN102)
 - all features of Python (→ <https://docs.python.org/>)
 - all programming concepts (→ IN102, IN103, IN104, IN204)

IN101 Modalites

- Main source of information is DFR website:
 - <http://www.dfr.ensta.fr/Cours/index.php?usebdd=ensta&sigle=IN101>
 - Schedule, links to books, links to PDFs of CM and TDs.

Cours DFR EN101: AL...

ENSTA ParisTech

IN101 - Algorithmique et Programmation (en Python)

Année	SE	Module	SE	CHARGES RELEVES
Professeurs :				
Freek STULP, Florian BANGREDEL, Alexander GEMPERT				
Maitres de conférences :				
Freek STULP, Alexander GEMPERT, Florian BANGREDEL, Thomas HOGOT, Siemans RAEDDA, Julien BLOCHER, EDU GABRIELLO, Aurélien CHEVALIER				
DATE	HEURES	TYPE	SEANCE	PROGRAMME DE LA SEANCE
05-09-2014	14h00-14h45	Cours magistral		Introduction, Python, variables, control flow (Freek)
06-09-2014	15h00-17h35	TD	en salle info	
07-09-2014	14h00-14h45	Cours magistral		Fonctions, ressource (Florian)
08-09-2014	15h00-17h35	TD	en salle info	
09-09-2014	14h00-14h45	Cours magistral		Arbres (Florian)
10-09-2014	15h00-17h35	TD	en salle info	
11-10-2014	14h00-14h45	Cours magistral		Searching and sorting (Alexander)
12-10-2014	15h00-17h35	TD	en salle info	
13-10-2014	14h00-14h45	Cours magistral		Advanced sorting (Alexander)
14-10-2014	15h00-17h35	TD	en salle info	
15-10-2014	14h00-14h45	Cours magistral		Classes, objects, linked lists (Freek)
16-10-2014	15h00-17h35	TD	en salle info	
17-11-2014	14h00-14h45	Cours magistral		Linked lists, trees (Freek)
18-11-2014	15h00-17h35	TD	en salle info	
19-11-2014	14h00-14h45	Cours magistral		Trees, summary, outlook (Florian)
20-11-2014	15h00-17h35	TD	en salle info	
21-11-2014	14h00-17h35	Corollaire		

IN101 Modalites

- Main source of information is DFR website:
 - <http://wwwdfr.ensta.fr/Cours/index.php?usebdd=ensta&sigle=IN101>
 - Schedule, links to books, links to PDFs of CM and TDs.
- Every week
 - 15:15 – $\approx$ 16:15 “cours magistral”
 - 15 minute break
 - $\approx$ 16:30 – 18:30 “travaux dirigés”
 - programming autonomously, exercises



The screenshot shows the IN101 website interface. On the left, there are navigation links: 'Enjeux', 'Méthodes', 'Pré-requis et débrouilles', 'Bibliographie, liens, supports', 'Contrôle de connaissances', 'Programmation situationnelle', and 'Autres liens à voir'. The main content area displays the course title 'IN101 - Algorithmique et Programmation (en Python)' and a table with course details.

ANNEE	SE	HEURE	CHAMBRE	TYPE	PROFESSEUR
2014	14	15:15	1404	Cours magistral	Freek STULP, Florian BANGREDDO, Alexander GÉPERTH
2014	15	16:15	1404	Cours magistral	Freek STULP, Alexander GÉPERTH, Florian BANGREDDO, Thomas HOGOT, Damien RAEDDA, Julien BLOCHET, ET GABRIELLO, Fabrice CHEVALER
2014	16	17:15	1404	Cours magistral	Introduction, Python, variables, control flow (Freek)
2014	17	18:15	1404	Cours magistral	Functions, recursion (Florian)
2014	18	19:15	1404	Cours magistral	Arrays (Florian)
2014	19	20:15	1404	Cours magistral	To en salle info
2014	20	21:15	1404	Cours magistral	searching and sorting (Alexander)
2014	21	22:15	1404	Cours magistral	advanced sorting (Alexander)
2014	22	23:15	1404	Cours magistral	classes, objects, linked lists (Freek)
2014	23	24:15	1404	Cours magistral	linked lists, trees (Freek)
2014	24	25:15	1404	Cours magistral	Trees, summary, outlook (Florian)
2014	25	26:15	1404	Cours magistral	Controlle

IN101 Modalites

- Main source of information is DFR website:
 - <http://www.dfr.ensta.fr/Cours/index.php?usebdd=ensta&sigle=IN101>
 - Schedule, links to books, links to PDFs of CM and TDs.
- Every week
 - 15:15 – ≈16:15 “cours magistral”
 - 15 minute break
 - ≈16:30 – 18:30 “travaux dirigés”
 - programming autonomously, exercises
- Contrôle de connaissances, marks
 - 50% 1 graded TD
 - 50% TD8, 15.11.2016



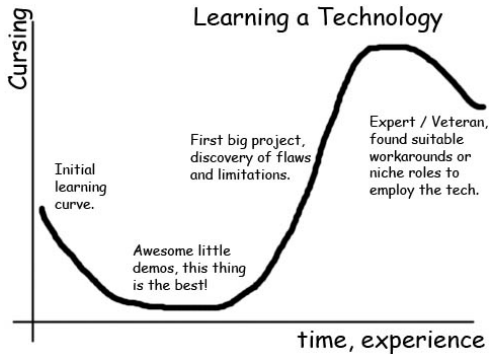
DATE	HEURE	TYPE	PROGRAMME DE LA SÉANCE
05.09.2014	14h00-14h45	Cours magistral	Introduction, Python, variables, control flow (Freek)
06.09.2014	15h00-17h30	Td en salle info	
23.09.2014	14h00-14h45	Cours magistral	fonctions, ressource (Freek)
30.09.2014	15h00-17h30	Td en salle info	
07.10.2014	14h00-14h45	Cours magistral	listes (Freek)
07.10.2014	15h00-17h30	Td en salle info	searching and sorting (Alexander)
14.10.2014	14h00-14h45	Cours magistral	advanced sorting (Alexander)
14.10.2014	15h00-17h30	Td en salle info	
21.10.2014	14h00-14h45	Cours magistral	classes, objects, linked lists (Freek)
21.10.2014	15h00-17h30	Td en salle info	
04.11.2014	14h00-14h45	Cours magistral	linked lists, trees (Freek)
04.11.2014	15h00-17h30	Td en salle info	
05.11.2014	14h00-14h45	Cours magistral	trees, summary, outlook (Freek)
05.11.2014	15h00-17h30	Td en salle info	
20.11.2014	14h00-17h30	Contrôle	

Who are we?

- Cours magistraux (+ travaux dirigés)
 - Vladimir Paun
 - Research topics: formal methods, hard real-time system verification, WCET
 - Nationality: Romanian
 - Email: paun@ensta-paristech.fr
 - Office: R.3.28 UIIS, ENSTA ParisTech

Who are you?

- Up until 2 years ago, 90% of the students that arrived at ENSTA had never programmed before
- But **you** have all (?) had Python in your “prépa”
 - but we don't know at all how well you learned it...



Who are you?

- Up until 2 years ago, 90% of the students that arrived at ENSTA had never programmed before
- But **you** have all (?) had Python in your “prépa”
 - but we don't know at all how well you learned it...
- Our strategy
 - If you don't know Python at all: we want you to be able to learn
 - “Basic” part of the TD
 - If you are already a hacker: we don't want to bore you
 - “Advanced” part of the TD (you can skip “Basic”, if you want)
 - “Everyone” part of the TD: for everyone
- Please give us your (constructive) feedback!

Outline

- 1 Algorithms
- 2 Programming
- 3 Cours IN101
- 4 Python Basics
 - Python Interpreter
 - Variables
 - Operators
 - Control flow
 - Ancient algorithms still used today
- 5 TD

Programming

- Programming: represent algorithms in a formal language that
 - Humans can (learn to) understand and write
 - Computers can read and execute
- Example languages: C, C++, Java, Matlab, Python

Programming

- Programming: represent algorithms in a formal language that
 - Humans can (learn to) understand and write
 - Computers can read and execute
- Example languages: C, C++, Java, Matlab, Python
- Why Python in IN101?
 - easy to learn
 - emphasizes readability
 - multiple programming paradigms
 - fun!
 - you already know it from the prépa

Origins of Python



Guido van Rossum – *“in December 1989, I was looking for a “hobby” programming project that would keep me occupied during the week around Christmas.”*

Name from *“Monty Python’s Flying Circus”*



Core philosophy: “The Zen of Python”

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



Origins of Python



Guido van Rossum – *“in December 1989, I was looking for a “hobby” programming project that would keep me occupied during the week around Christmas.”*

Name from *“Monty Python’s Flying Circus”*



Core philosophy: “The Zen of Python”

- Préférer le beau au laid,
- l'explicite à l'implicite,
- le simple au complexe,
- le complexe au compliqué,



And now for something completely different...



let's write some programs in Python!

“Hello World” in the Python language

- Put the following in a file `helloworld.py`
 - Called a “Python script”

```
print("Hello World!")
```

“Hello World” in the Python language

- Put the following in a file `helloworld.py`
 - Called a “Python script”

```
print("Hello World!")
```

- Execute the instructions in the script in a terminal
 - Calls the “Python interpreter”
 - Interprets the instructions and executes them one by one
 - “Reads” from top to bottom

```
commlne> python3 helloworld.py  
Hello World!
```

Comments in Python

- Comments
 - Everything on a line following `#` (le dièze)
 - Ignored by the Python interpreter
 - Intended only for humans reading the script
- Why comments?
 - Documentation: explaining the code

```
# This is a Python script that will print  
# "Hello World"  
print("Hello World!")
```

```
comline> python3 helloworldcomments.py  
Hello World!
```

Summary

- Python is a programming language
- The Python interpreter...
 - is called from the command line (in a terminal)
 - interprets a Python script from top to bottom
 - executes the instructions in the script
 - ignore comments (everything after a #)

Summary

- Python is a programming language
- The Python interpreter...
 - is called from the command line (in a terminal)
 - interprets a Python script from top to bottom
 - executes the instructions in the script
 - ignore comments (everything after a #)
- Writing a Python script

Outline

- 1 Algorithms
- 2 Programming
- 3 Cours IN101
- 4 Python Basics
 - Python Interpreter
 - Variables
 - Operators
 - Control flow
 - Ancient algorithms still used today
- 5 TD

Variables

- Variables in mathematics
 - Character which is not fully specified, or unknown
 - Example: $f(x) = x^2$ (what is the value of the variable x ?)
- Variables in computer science and programming are different. . .

Variables: names and values

- Variables (in computer science/programming)
 - A name/identifier (“a”) associated with a value (“1”)
 - Assignment: associating a variable with a value (“a = 1”)
 - The value of a variable can change



```
a = 1      # Assign value '1' to variable with name 'a'  
print(a)
```

```
commline> python3 variables1.py  
1
```

Variables: names and values

- Variables (in computer science/programming)
 - A name/identifier ("a") associated with a value ("1")
 - Assignment: associating a variable with a value ("a = 1")
 - The value of a variable can change



```
a = 1      # Assign value '1' to variable with name 'a'
print(a)
a = 7      # Assign a new value to variable 'a'
print(a)
```

```
commline> python3 variables1.py
1
7
```

Variables: names and values

- Variables (in computer science/programming)
 - A name/identifier ("a") associated with a value ("1")
 - Assignment: associating a variable with a value ("a = 1")
 - The value of a variable can change



```
a = 1      # Assign value '1' to variable with name 'a'
print(a)
a = 7      # Assign a new value to variable 'a'
print(a)
b = 3      # Assign value '3' to variable with name 'b'
print(b)
```

```
commline> python3 variables1.py
1
7
3
```

Variables: names and values

- Variables (in computer science/programming)
 - A name/identifier ("a") associated with a value ("1")
 - Assignment: associating a variable with a value ("a = 1")
 - The value of a variable can change



```
a = 1      # Assign value '1' to variable with name 'a'
print(a)
a = 7      # Assign a new value to variable 'a'
print(a)
b = 3      # Assign value '3' to variable with name 'b'
print(b)
b = a * b  # Assign value 'a*b' (=7*3) to variable with name 'b'
print(b)
```

```
commline> python3 variables1.py
1
7
3
21
```

Different types of variables

- What types of variables are there? Some examples

Numbers – whole numbers or real numbers

Boolean – either `True` or `False`

Strings – sequence of characters

```
whole_number = 1
real_number   = 1.540
boolean       = True
string        = "Hello world"
print(whole_number)
print(real_number)
print(boolean)
print(string)
```

```
commlne> python3 variables2.py
1
1.54
True
Hello world
```

Different types of variables

- What types of variables are there? Some examples

Numbers – whole numbers or real numbers

Boolean – either `True` or `False`

Strings – sequence of characters

- Determine variable type with `type()`

```
whole_number = 1
real_number  = 1.540
boolean      = True
string       = "Hello world"
print(type(whole_number)) # integer
print(type(real_number))  # float
print(type(boolean))      # bool
print(type(string))       # str
```

```
commlne> python3 variables3.py
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

Different types of variables

- What types of variables are there? Some examples

Numbers – whole numbers or real numbers

Boolean – either `True` or `False`

Strings – sequence of characters

- Determine variable type with `type()`
- 'Typing' discussed in detail in IN102

```
whole_number = 1
real_number  = 1.540
boolean      = True
string       = "Hello world"
print(type(whole_number))  # integer
print(type(real_number))   # float
print(type(boolean))       # bool
print(type(string))        # str
```

```
commlne> python3 variables3.py
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`

```
a = 1  # Assignment operator
print(a)
a = 3  # Assignment operator
print(a)
```

```
commlne> python3 operators1.py
1
3
```

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`
 - Arithmetic Operators: `+` `-` `*` `/` `**` `%`

```
a = 4 + 3          # Addition
print(a)
print( 4 - 3 )     # Subtraction
print( 4 * 3 )     # Multiplication
print( 4 / 3 )     # Division
print( 4 ** 3 )    # To the power of
```

```
commlne> python3 operators2.py
7
1
12
1.3333333333333333
64
```

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`
 - Arithmetic Operators: `+` `-` `*` `/` `**` `%`

```
# Division
print( 9 / 2 ) # => 4.5 (float)
print( 9 // 2 ) # => 4 (int)

# Remainder (modulo operator)
print( 9 % 9 )
print( 9 % 7 )
```

```
commlne> python3 operators4.py
4.5
4
0
2
```

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`
 - Arithmetic Operators: `+` `-` `*` `/` `**` `%`
 - Comparison Operators: `==` `!=` `<` `>` `<=` `>=`

```
print( 4 == 3 ) # Equal?
print( 4 != 3 ) # Not equal?
print( 4 < 3 )  # Smaller than?
print( 4 > 3 )  # Larger than?

a = (4==3) # Assign result of operator
           # to variable with name 'a'
print(a)
```

```
commlne> python3 operators3.py
False
True
False
True
False
```

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`
 - Arithmetic Operators: `+` `-` `*` `/` `**` `%`
 - Comparison Operators: `==` `!=` `<` `>` `<=` `>=`
 - Logical Operators: `and` `or` `not`

```
print( 1>0 and 1>2 )      # Both true: No...
print( 1>0 or 1>2 )       # One of them true: Yes!
print( not (1>0 and 1>2) ) # Not true that both are true? Yes!
print( not not 1>0 )      # Double negation
```

```
commlne> python3 operators5.py
False
True
True
True
```

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`
 - Arithmetic Operators: `+` `-` `*` `/` `**` `%`
 - Comparison Operators: `==` `!=` `<` `>` `<=` `>=`
 - Logical Operators: `and` `or` `not`
- Expression: valid combination of value, variables and operators
 - Examples: `23` `23+4` `23+a` `2*(23+a)`
 - Examples: `True` `True and (1>0)`

Operators

- Returns a value when evaluated (e.g. $4 + 3$ evaluates to 7)
 - Assignment Operators: `=`
 - Arithmetic Operators: `+` `-` `*` `/` `**` `%`
 - Comparison Operators: `==` `!=` `<` `>` `<=` `>=`
 - Logical Operators: `and` `or` `not`
- Later in IN101
 - Formal definition
 - Membership and Identity Operators
 - `+=` `-=` `*=` `/=` `**=`
 - Operators are actually functions (CM02)

Summary

- Variables are names associated with values
- Variables and values may be combined with operators
 - this yields expressions, which themselves yield values

Outline

- 1 Algorithms
- 2 Programming
- 3 Cours IN101
- 4 Python Basics
 - Python Interpreter
 - Variables
 - Operators
 - Control flow
 - Ancient algorithms still used today
- 5 TD

Outline: Control Flow

- 1 for loop (“boucle for”)
- 2 conditional execution (“alternatives”)
- 3 while loop (“boucle tant que”)

Control Flow: for loop (“boucle for”)

- What if you want to execute the same instruction several times?

```
1 number = 0
2 print(7 * number)
3 number = 1
4 print(7 * number)
5 number = 2
6 print(7 * number)
7 number = 3
8 print(7 * number)
```

```
commlne> python3 forloop0.py
0
7
14
21
```

Control Flow: for loop (“boucle for”)

- What if you want to execute the same instruction several times?
- for loop: assigns values in a list to a variable, one after the other

```
1 number = 0
2 print(7 * number)
3 number = 1
4 print(7 * number)
5 number = 2
6 print(7 * number)
7 number = 3
8 print(7 * number)
```

```
for number in [0, 1, 2, 3]:
    print(7 * number)
```

```
commlne> python3 forloop0.py
0
7
14
21
```

```
commlne> python3 forloop1.py
0
7
14
21
```

Control Flow: for loop (“boucle for”)

- What if you want to execute the same instruction several times?
- for loop: assigns values in a list to a variable, one after the other
- range() function (→ CM2) generates sequences of numbers
 - Careful, range(n1,n2) stops at n2-1
 - range(n) starts at 0 and stops at n-1

```
1 number = 0
2 print(7 * number)
3 number = 1
4 print(7 * number)
5 number = 2
6 print(7 * number)
7 number = 3
8 print(7 * number)
```

```
for number in [0, 1, 2, 3]:
    print(7 * number)

for number in range(0, 4):
    print(7 * number)

for number in range(4):
    print(7 * number)
```

```
commline> python3 forloop0.py
0
7
14
21
```

```
commline> python3 forloop2.py
0
7
14
21
0
```

Indentation and code blocks

- Indentation is very important in Python

Indentation and code blocks

- Indentation is very important in Python

```
1 for number in [0, 1, 2]:  
2  
3     # Inside the code block  
4     print(7 * number)  
5  
6     # Inside the code block  
7     print("abc")
```

```
for number in [0, 1, 2]:  
  
    # Inside the code block  
    print(7 * number)  
  
    # Outside the code block  
    print("abc")
```

```
commlne> python3 indentation1.py  
0  
abc  
7  
abc  
14
```

```
commlne> python3 indentation2.py  
0  
7  
14  
abc
```

Indentation and code blocks

- Indentation is very important in Python
- Code block (“bloc d’instructions”)
 - A section of code which is grouped together.
 - Python groups code based on indentation

```
1 for number in [0, 1, 2]:  
2  
3     # Inside the code block  
4     print(7 * number)  
5  
6     # Inside the code block  
7     print("abc")
```

```
for number in [0, 1, 2]:  
  
    # Inside the code block  
    print(7 * number)  
  
# Outside the code block  
print("abc")
```

```
commlne> python3 indentation1.py  
0  
abc  
7  
abc  
14
```

```
commlne> python3 indentation2.py  
0  
7  
14  
abc
```

Indentation and code blocks

- Indentation is very important in Python
- Code block (“bloc d’instructions”)
 - A section of code which is grouped together.
 - Python groups code based on indentation
- Indentation: 4 spaces

```
1 for number in [0, 1, 2]:  
2  
3     # Inside the code block  
4     print(7 * number)  
5  
6     # Inside the code block  
7     print("abc")
```

```
for number in [0, 1, 2]:  
  
    # Inside the code block  
    print(7 * number)  
  
    # Outside the code block  
    print("abc")
```

```
commlne> python3 indentation1.py  
0  
abc  
7  
abc  
14
```

```
commlne> python3 indentation2.py  
0  
7  
14  
abc
```

Nested loop (“boucle imbriquée”)

- Nested loop: a loop inside a loop

```
for number1 in [4000, 1000]:  
    for number2 in [20, 30]:  
        print(number1 + number2)
```

```
commline> python3 forloopnested.py  
4020  
4030  
1020  
1030
```

Nested loop (“boucle imbriquée”)

- Nested loop: a loop inside a loop

```
for number1 in [4000, 1000]:  
    for number2 in [20, 30]:  
        for number3 in [6, 7]:  
            print(number1 + number2 + number3)
```

```
commline> python3 forloopnested2.py  
4026  
4027  
4036  
4037  
1026  
1027  
1036  
1037
```

A first algorithm with for: factorial

- Implement factorial: $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - Basic version (only for $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)

```
# Algorithm: computes factorial = 4! = 1*2*3*4 = 24
factorial = 1           # 1
factorial = 2 * factorial # 2 <- 2*1
factorial = 3 * factorial # 6 <- 3*2
factorial = 4 * factorial # 24 <- 4*6

print(factorial) # Output
```

```
commlne> python3 factorial0.py
24
```

A first algorithm with for: factorial

- Implement factorial: $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - Basic version (only for $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - With for loop (only for $f(4)$)

```
# Algorithm: computes factorial = 4! = 1*2*3*4 = 24
factorial = 1
for i in [2, 3, 4]:                # 2, 3, 4
    factorial = i * factorial      # 2<-2*1, 6<-3*2, 24<-4*6

print(factorial) # Output
```

```
commlne> python3 factorial1.py
24
```

A first algorithm with for: factorial

- Implement factorial: $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - Basic version (only for $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - With for loop (only for $f(4)$)
 - With range (only for $f(4)$)

```
# Algorithm: computes factorial = 4! = 1*2*3*4 = 24
factorial = 1
for i in range(2, 4+1):      # 2, 3, 4
    factorial = i * factorial # 2<-2*1, 6<-3*2, 24<-4*6

print(factorial) # Output
```

```
commlne> python3 factorial1a.py
24
```

A first algorithm with for: factorial

- Implement factorial: $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - Basic version (only for $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - With for loop (only for $f(4)$)
 - With range (only for $f(4)$)
 - With input (for $f(n)$)

```
n = 4 # Input
# Algorithm: computes factorial = n! = 1*2*...*(n-2)*(n-1)*n
factorial = 1
for i in range(2, n+1):      # 2, 3, 4, ..., n-2, n-1, n
    factorial = i * factorial # 2=2*1, 6=3*2, 24=4*6, etc.

print(factorial) # Output
```

```
commlne> python3 factorial2.py
24
```

A first algorithm with for: factorial

- Implement factorial: $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - Basic version (only for $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - With for loop (only for $f(4)$)
 - With range (only for $f(4)$)
 - With input (for $f(n)$)

```
n = 10 # Input
# Algorithm: computes factorial = n! = 1*2*...*(n-2)*(n-1)*n
factorial = 1
for i in range(2, n+1):      # 2, 3, 4, ..., n-2, n-1, n
    factorial = i * factorial # 2=2*1, 6=3*2, 24=4*6, etc.

print(factorial) # Output
```

```
comline> python3 factorial2a.py
3628800
```

Conditional execution (“alternatives”)

- if statement (“test si”)
 - execute instructions only if a certain condition holds

```
a = 3
b = 4
if (a < b): # Condition (expression yielding boolean)
    print("a is smaller than b")
if (b <= a): # Condition (expression yielding boolean)
    print("b is smaller than or equal to a")
```

```
commlne> python3 ifstatement.py
a is smaller than b
```

Conditional execution (“alternatives”)

- if statement (“test si”)
 - execute instructions only if a certain condition holds
- if-then-else statement (“test si sinon”): for convenience

```
a = 3
b = 4
if (a < b): # Condition (expression yielding boolean)
    print("a is smaller than b")
else: # If condition above doesn't hold
    print("b is smaller than or equal to a")
```

```
comline> python3 ifthenelsestatement.py
a is smaller than b
```

Conditional execution (“alternatives”)

- if statement (“test si”)
 - execute instructions only if a certain condition holds
- if-then-else statement (“test si sinon”): for convenience
- elif: combines else and if: for convenience

```
a = 3
b = 4
if (a < b):  # Condition (expression yielding boolean)
    print("a is smaller than b")
elif (a > b):  # Condition (expression yielding boolean)
    print("a is larger than b")
else:        # If conditions above do not hold
    print("a is equal to b")
```

```
comline> python3 ifthenelifelsestatement.py
a is smaller than b
```

Control Flow: while loop (“boucle tant que”)

- Combines a loop with a condition
 - useful when you don't know number of iterations in advance

```
# Print(the first multiple of 7)  
# that is larger than 10000  
number = 0  
while (number <= 10000):  
    number = number + 7  
print(number)
```

```
comline> python3 whileloop.py  
10003
```

Outline

- 1 Algorithms
- 2 Programming
- 3 Cours IN101
- 4 Python Basics
 - Python Interpreter
 - Variables
 - Operators
 - Control flow
 - Ancient algorithms still used today
- 5 TD

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; that value is the greatest common divisor of the original pair. (taken from Wikipedia)

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; that value is the greatest common divisor of the original pair. (taken from Wikipedia)
- **Programming:** Euclid's algorithm in Python

```
a = 12 # Input  
b = 16 # Input
```

Euclid's algorithm (≈ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; that value is the greatest common divisor of the original pair. (taken from Wikipedia)
- **Programming:** Euclid's algorithm in Python

```
a = 12 # Input
b = 16 # Input

if (a > b):
    a = a - b # 'a' is the largest number
else:
    b = b - a # 'b' is the largest number
```

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; that value is the greatest common divisor of the original pair. (taken from Wikipedia)
- **Programming:** Euclid's algorithm in Python

```
a = 12 # Input
b = 16 # Input

while (???):    # continue until...
    if (a > b):
        a = a - b # 'a' is the largest number
    else:
        b = b - a # 'b' is the largest number
```

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until **the numbers in the new pair are equal to each other**; that value is the greatest common divisor of the original pair. (taken from Wikipedia)
- **Programming:** Euclid's algorithm in Python

```
a = 12 # Input
b = 16 # Input

while (a != b): # continue until 'a' and 'b' are equal
    if (a > b):
        a = a - b # 'a' is the largest number
    else:
        b = b - a # 'b' is the largest number
```

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; **that value is the greatest common divisor of the original pair.** (taken from Wikipedia)
- **Programming:** Euclid's algorithm in Python

```
a = 12 # Input
b = 16 # Input

while (a != b): # continue until 'a' and 'b' are equal
    if (a > b):
        a = a - b # 'a' is the largest number
    else:
        b = b - a # 'b' is the largest number

gcd = a
print(gcd) # Output
```

Euclid's algorithm ($\approx$ 300 BC)

- **Specification:** Greatest common divisor ("Plus grand commun diviseur")
 - Divisors of 12: 1, 2, 3, 4, 6, 12
 - Divisors of 16: 1, 2, 4, 8, 16
 - Greatest common divisor of 12 and 16: 4
- **Algorithm:** Euclid's algorithm in English
 - Starting with a pair of positive integers, form a new pair consisting of the smaller number and the difference between the larger number and the smaller number. This process repeats until the numbers in the new pair are equal to each other; that value is the greatest common divisor of the original pair. (taken from Wikipedia)
- Why is a > 2 millenia old algorithm still important today?
 - Used in security protocols (e.g. in the RSA algorithm)
 - Any secure communication (e.g. <https://>) will call this algorithm many times

Sieve of Eratosthenes ($\approx$ 200 BC)

- Prime number (“nombre premier”)
 - natural number $\{1, 2, 3, \dots\}$
 - greater than 1
 - only 2 positive divisors: 1 and itself
- Requested algorithm
 - Input: number n
 - Output: all prime numbers $\leq n$

Sieve of Eratosthenes ($\approx$ 200 BC)

- Prime number (“nombre premier”)
 - natural number $\{1, 2, 3, \dots\}$
 - greater than 1
 - only 2 positive divisors: 1 and itself
- Requested algorithm
 - Input: number n
 - Output: all prime numbers $\leq n$

Design an algorithm for this with your neighbor(s)!

Sieve of Eratosthenes ($\approx$ 200 BC)

- Prime number (“nombre premier”)
 - natural number $\{1, 2, 3, \dots\}$
 - greater than 1
 - only 2 positive divisors: 1 and itself
- Requested algorithm
 - Input: number n
 - Output: all prime numbers $\leq n$

Design an algorithm for this with your neighbor(s)!

Sieve of Eratosthenes

- Eratosthenes of Cyrene, 276 BC – 195 BC
- Found one of the most efficient algorithms
- Explanation will have to wait until CM3...



Prime factorization

- Requested algorithm
 - Input: n , the product of two unknown primes $n = a \cdot b$
 - Output: a and b

Prime factorization

- Requested algorithm
 - Input: n , the product of two unknown primes $n = a \cdot b$
 - Output: a and b
- Finding a slow algorithm is straightforward (TD2)
 - 1 Compute all primes up to $n - 2$
 - Using the algorithm you just made
 - Or the Sieve of Eratosthenes
 - 2 Multiply combinations of primes until you get n

Prime factorization

- Requested algorithm
 - Input: n , the product of two unknown primes $n = a \cdot b$
 - Output: a and b
- Finding a slow algorithm is straightforward (TD2)
 - 1 Compute all primes up to $n - 2$
 - Using the algorithm you just made
 - Or the Sieve of Eratosthenes
 - 2 Multiply combinations of primes until you get n
- Finding a fast algorithm is *really* difficult

Prime factorization

- Requested algorithm
 - Input: n , the product of two unknown primes $n = a \cdot b$
 - Output: a and b
- Finding a slow algorithm is straightforward (TD2)
 - 1 Compute all primes up to $n - 2$
 - Using the algorithm you just made
 - Or the Sieve of Eratosthenes
 - 2 Multiply combinations of primes until you get n
- Finding a fast algorithm is *really* difficult
 - if you find a fast algorithm, I will pay you 1.000.000EUR for it.

Summary of this CM

- Algorithm
 - a recipe with input, instructions, and output
 - a set of rules that precisely defines a sequence of operations.
- Programming
 - implement algorithms in a (formal) language that can be written by a human, and executed by a computer
- Python
 - programming language, easy to learn, fun, Monty Python
 - executing instructions in a script: Python interpreter
- Python language
 - variables: names with values
 - types of values: numbers, lists, strings, booleans
 - operators: combining values and variables
 - control flow: for, if, while

Aims of the upcoming TD

- Write your first Python program, calling the interpreter
- Design your first algorithm
- Program your first algorithms
- Create a clean, comfortable programming environment
 - Choose an editor you are proficient in
- Learn to find solutions yourself, with help from
 - the slides
 - the Python documentation: <https://docs.python.org/3/>
 - your favourite search engine
 - the books that are provided as PDFs

Part I: Basics

CM1 variables, control flow

a=1, for/if/while

CM2 functions and modules

factorial(6), import math

Part II: Data Structures (Built-in)

CM3 lists, arrays, dictionaries

[2, 3, 1]

Part III: Search and Sort

CM4 searching/sorting

[2, 3, 1] → [1, 2, 3]

CM5 Exam 1

Part IV: Linked Data Structures

CM6 objects/classes

circle_object = Circle(0,1,3)

CM7 linked lists

'a' → 'e' → 'b' → x

CM8 trees

ψ

CM9 Exam 2

Have fun programming your first algorithms in the first TD!

