

1. Tout programme peut être vu comme une donnée

La plupart des programmes **attendent en entrée des données (input)** pour faire ce qu'on attend d'eux, et produire en sortie un résultat (**output**).

Exemple :

```
1 def prog(t):
2     m=t[0]
3     for e in t:
4         if e < m:
5             m=e
6     return m
```



Lors de l'appel à la fonction **prog** :

- Quel type d'entrée sera attendu ? **liste**
- Quelle sortie sera renvoyée ? **Un élément du dictionnaire (chaîne de caractères, float, int, bool, liste, dico)**

Ces données d'entrée peuvent être de types très divers :

- Types de base : **chaîne de caractères, float, int, bool**
- Ou types construits : **liste, dico**

Cela peut également être des fichiers textes, ou n'importe quel type de fichier, en particulier : il est tout à fait possible **d'avoir en donnée d'entrée**



a. Exemple de l'interpréteur Python

Quand on enregistre un programme Python dans un fichier, par exemple **test_prog.py**, et qu'on exécute ce dernier avec Pyzo, que se passe-t-il derrière ?

```
1 def prog(t):
2     m=t[0]
3     for e in t:
4         if e < m:
5             m=e
6     return m
7
8 print(prog([2,5,1,9]))
```

```
Python 3.7.6 (default, Jan 8 2020, 20:23:39) on Windows (64 bits).
This is the Pyzo interpreter with integrated event loop for ASYNCIO.
Type 'help' for help, type '?' for a list of *magic* commands.
Running script: "C:\Users\marin\test_prog.py"
1
>>>
```



On le voit encore mieux quand on appelle l'interpréteur Python directement depuis une fenêtre de commande système, au lieu de passer par Pyzo : il faut lui passer le nom du programme en argument.

```
(base) C:\Users\marin>
(base) C:\Users\marin>cd Miniconda3
(base) C:\Users\marin\Miniconda3>python C:\Users\marin\test_prog.py
```

1

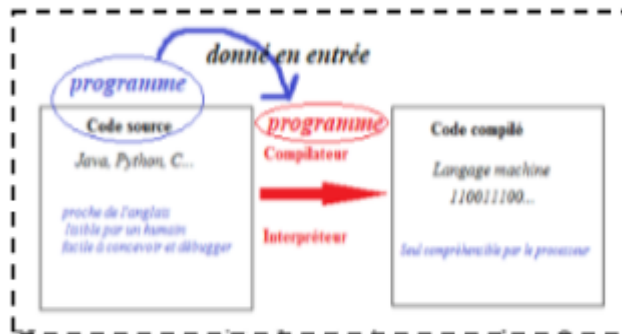
résultat en sortie

interpréteur python

programme passé en entrée

b. Exemple d'un compilateur (langage C)

RAPPEL DE 1ÈRE :



Les ordinateurs ne comprennent que le langage machine, c'est-à-dire des 0 et des 1, alors que le programmeur préfère utiliser un langage de « haut niveau », proche de l'anglais naturel. On doit donc utiliser un programme qui transforme ce langage évolué (Python, C, Java, C++...) en langage machine. Ce programme est un **interpréteur**, ou un **compilateur**.

Un programme existe donc toujours sous deux formes : le **code source** et le **code compile**

Quand on programme par exemple en langage C :

- Une fois le code-source terminé, on passe le programme en entrée au compilateur (qui est lui-même un logiciel, donc un programme)
- Ce compilateur traite le code-source donné en entrée
- ... et renvoie en sortie un fichier contenant le code compilé, c'est-à-dire transformé en langage binaire compréhensible par la machine.

C'est une **notion fondamentale** qui transparaît dans ces exemples : ...

.....

.....

c. Autres exemples où la donnée d'entrée est un programme

- ❑ **Quand on télécharge un logiciel** sur Internet : on utilise un gestionnaire de téléchargement qui est lui-même un logiciel, donc un programme.
- ❑ **Un système d'exploitation** est un « programme géant » qui fait tourner plein de programmes différents

Anecdote : le 4 juin 1996, le vol inaugural du lanceur européen Ariane 5 s'est soldé par un échec causé par un dysfonctionnement informatique : la fusée a explosé en vol seulement 36.7 secondes après le décollage. Depuis cette date, la France a développé de nombreux programmes permettant de valider d'autres programmes

Détection d'erreurs :

- ❑ La plupart des éditeurs de programmation possèdent des « **détecteurs d'erreur** » pour le **code source**.
- ❑ Il existe même des programmes qui peuvent prouver mathématiquement qu'un autre programme fait bien ce pour quoi il a été conçu...

Exercice 1

Ecrire le script d'une fonction Python `nb_lignes(fich)` qui renvoie le nombre de lignes du programme contenu dans le fichier `fich`



2. Le problème de l'arrêt

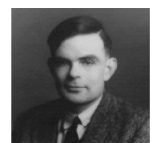
Le "cauchemar" d'un programmeur est que son programme, dans certains cas, "tombe" dans une boucle infinie et ne s'arrête jamais. Dans ce cas, le logiciel est incapable de fournir la réponse attendue par l'utilisateur.

- ⇒ Un programme qui permettrait de tester si un autre programme va finir par s'arrêter, quel que soit le cas traité, serait d'une grande aide pour tous les développeurs du monde !



Alonzo Church (1903-1995)

Pourtant, depuis 1937 et les travaux d'Alonzo Church et d'Alan Turing, on sait qu'un tel programme ne peut pas exister : on dit que



Alan Turing (1912-1954)

Voir vidéo :) : <https://youtu.be/13O1qhX4Bqo>
<https://www.youtube.com/watch?v=92WHN-pAFCs>

Démonstration :

Par l'absurde : on suppose qu'il existe un tel programme permettant de prédire l'arrêt

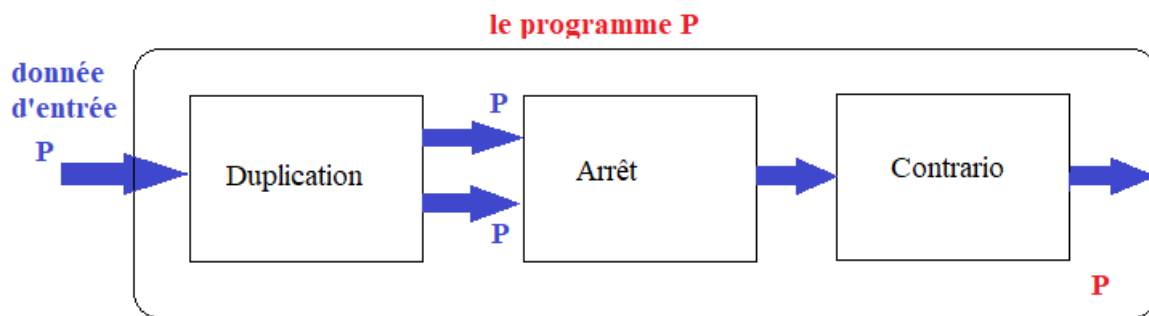


On considère alors deux autres programmes simples :

- ⇒ Duplication (une « photocopieuse »)
- ⇒ Contrario : qui part dans une boucle infinie ou renvoie 1 selon qu'on lui donne 1 ou pas en entrée

On construit alors à partir de ces trois briques un programme nommé **P** qui va mener à une absurdité logique lorsqu'on l'appliquera à lui-même.

En effet, on a vu que tout programme peut être considéré comme une donnée d'entrée : on peut donc appliquer un programme à lui-même, comme ci-dessous :



Regardons alors les différents cas possibles :

Premier cas : si **P** s'arrête quand on lui passe **P** en entrée : alors

.....

Deuxième cas : si **P** ne s'arrête pas quand on lui passe **P** en entrée : alors

.....

On a envisagé tous les cas possibles, et ils mènent chacun à une absurdité logique. C'est donc qu'on a fait une hypothèse fausse au départ du raisonnement : en réalité, le programme « Arrêt » ne peut pas exister, c'est impossible.

A RETENIR : il n'existe pas de programme « universel » qui pourrait déterminer dans le cas général si un programme se termine sur une entrée donnée.

Cette idée d'avoir un problème qui ne peut pas être résolu par un algorithme peut être généralisée :

DÉFINITION : on dit qu'un problème est décidable

....., et qu'il est indécidable dans le cas contraire.


3. Calculabilité, machine de Turing

a. Fonctions calculables

Dans le même esprit, on distingue les **fonctions calculables** de celles qui ne le sont pas.

De nombreuses fonctions mathématiques peuvent être décrites par une suite de manipulations définies à l'aide de symboles, et visualisées mentalement comme des déplacements de petits cailloux. Ces fonctions ont été identifiées et classifiées au début des années 1930 par l'américain Alonzo Church.

« Calcul » vient du latin « calculus » qui signifie « caillou ». Par exemple, on peut matérialiser concrètement les additions, soustractions, multiplications et divisions par des manipulations sur des petits cailloux : ce sont des fonctions calculables.

$$3 + 2 = 5$$


En revanche, d'autres fonctions plus compliquées sont impossibles à représenter ainsi : on ne peut pas **calculer** $f(x)$ à l'aide d'un algorithme.

ATTENTION : on peut arriver à trouver la valeur de $f(x)$, mais pas en **appliquant un algorithme**, c'est-à-dire un calcul au sens de Church.

Culture générale : chercher sur Internet l'exemple du « castor affairé »

EXEMPLE DE FONCTION NON CALCULABLE :

b. Machines de Turing

Le britannique Alan Turing a prouvé en 1937 que **les fonctions calculables au sens de Church étaient exactement les fonctions programmables sur la machine imaginaire qu'il avait conçue.**

Vidéos à regarder : https://videotheque.cnrs.fr/index.php?urlaction=doc&id_doc=3001 jusqu'à 2:36

Puis : <https://www.youtube.com/watch?v=P66h8D5Lkwk> jusqu'à 4:32

DE QUOI EST CONSTITUÉE UNE MACHINE DE TURING ?

A RETENIR : une fonction est dite calculable lorsqu'on peut la programmer grâce à un algorithme :

- ☐ consistant en un **ensemble fini d'instructions simples** et précises, décrites avec un **nombre limité de symboles**
- ☐ produisant le résultat en un **nombre fini d'étapes**.
- ☐ pouvant être suivi par un humain avec seulement du papier et un crayon
- ☐ l'exécution ne requérant aucune « d'intelligence » autre que la compréhension des instructions

Tout programme d'ordinateur, peu importe le langage de programmation, peut être traduit en une machine de Turing. On parle donc de **machine universelle**.

c. Exercices d'application

Exercice 1

Considérons une machine de Turing se promenant sur un ruban constitué d'une suite de cases pouvant être vierges (« V ») ou contenir une information binaire (0 ou 1).

Cette machine possède la faculté de se déplacer d'une case vers la gauche ou vers la droite, de lire ou d'écrire un caractère inscrit



Yves Coudert - Lycée Le Bon Sauve
D'après un document de Marine Méra

État	Caractère lu	Écrire	Déplacement	Nouvel état
E ₀	V	V	droite	E ₀
	1	0	droite	E ₁
	0	1	droite	E ₁
E ₁	V	Arrêt		
	1	0	droite	E ₁
	0	1	droite	E ₁

sur le ruban (dans ce cas, elle remplace le caractère qui y était écrit). De plus, elle possède un certain nombre d'états qui vont déterminer son comportement, et qui sont décrits dans la table de transition ci-contre.

Au début, la tête de lecture/écriture est positionnée dans l'état E_0 sur une des cases vierges située à gauche du nombre binaire.

- a. Faire fonctionner cette machine de Turing sur le ruban suivant :

...	V	V	0	1	0	0	1	1	V	V	...
-----	---	---	---	---	---	---	---	---	---	---	-----

- b. Quelle est l'opération effectuée par cette machine de Turing ?

Exercice 2

On considère une machine de Turing avec 9 états dont la table de transition est donnée page suivante.

Au départ, la tête de lecture/écriture est positionnée dans l'état E_0 sur le symbole binaire de gauche du nombre inscrit sur le ruban. Comme dans l'exercice précédent, le ruban contient une suite continue de symboles binaires, les autres cases étant vierges.

État	Caractère lu	Écrire	Déplacement	Nouvel état
E_0	V	Arrêt		
	0	V	droite	E_1
	1	V	droite	E_5
E_1	V	V	droite	E_2
	0-1	inchangé	droite	E_1
E_2	V	0	gauche	E_3
	0-1	inchangé	droite	E_2
E_3	V	V	gauche	E_4
	0-1	inchangé	gauche	E_3

État	Caractère lu	Écrire	Déplacement	Nouvel état
E_4	V	0	droite	E_0
	0-1	inchangé	gauche	E_4
E_5	V	V	droite	E_6
	0-1	inchangé	droite	E_5
E_6	V	1	gauche	E_7
	0-1	inchangé	droite	E_6
E_7	V	V	gauche	E_8
	0-1	inchangé	gauche	E_7
E_8	V	1	droite	E_0
	0-1	inchangé	gauche	E_8

- a. Faire fonctionner cette machine de Turing sur le ruban suivant (compléter les cases du tableau au fur et à mesure)

...	V	E_0	1	1	0	V	V	V	V	V	...
-----	---	-------	---	---	---	---	---	---	---	---	-----

fin											

- b. Quelle est l'opération effectuée par cette machine de Turing ?
- c. Que faudrait-il changer dans l'algorithme pour que le nombre inscrit sur le ruban soit recopié avec une « inversion » des symboles binaires ?
c'est-à-dire que le ruban V01101VVV... devienne V01101V10010VV...

Exercice 3

On considère une machine de Turing dont l'alphabet comporte 3 symboles : $\{-; 0; 1\}$ et qui possède plusieurs états nommés q_1, q_2, q_3, \dots

Pour définir la machine, on décrit sa table de transition, qui se présente sous la forme d'un tableau d'instructions, comme dans l'exemple ci-contre :

La machine s'arrête lorsqu'elle se trouve dans une configuration pour laquelle il n'y a pas d'instruction.


État init.	Lu	Écrit	Dir.	État suiv.
q_1	0	0	←	q_1
q_1	1	1	←	q_1
q_1	-	-	→	q_2
q_2	0	1	→	q_3
q_2	1	0	→	q_3

Préliminaire : appliquer cette table au ruban suivant

-	1	0	1	
			\hat{q}_1	

Montrer que l'état final de la machine est :

-	0	0	1	
---	---	---	---	--

	Turing décidable	Décidabilité
---	------------------	--------------

On dit alors que cette machine permet de « résoudre le problème » :

-	1	0	1	
			$\widehat{q_1}$	
-	0	0	1	

A vous de jouer :

Pour chaque problème présenté, vous devez donner les instructions permettant de le résoudre, c'est-à-dire construire la table de transition appropriée.

Vous avez à chaque fois la description de ce que vous devez faire et l'alphabet à utiliser.

Vous ne pouvez pas utiliser d'autres symboles. Vous avez aussi une indication du nombre d'états à utiliser, mais vous pouvez en utiliser plus.

La machine commence toujours dans l'état q_1 à la position indiquée dans chaque exemple. L'objectif est d'obtenir la bande du bas !

1 – Début Nb d'états : 1 Alphabet : {0;1;-} Description : Écrire un 1. Il n'y a pas besoin de s'inquiéter du déplacement sur la bande.	<table><tr><td></td><td>-</td><td></td></tr><tr><td></td><td>$\widehat{q_1}$</td><td></td></tr><tr><td></td><td>1</td><td></td></tr></table>		-			$\widehat{q_1}$			1																																
	-																																								
	$\widehat{q_1}$																																								
	1																																								
2 – À droite Nb d'états : 1 Alphabet : {0;1;-} Description : Il suffit d'une seule instruction.	<table><tr><td></td><td>-</td><td>-</td><td></td></tr><tr><td></td><td>$\widehat{q_1}$</td><td>↓</td><td></td></tr><tr><td></td><td>1</td><td>1</td><td></td></tr></table>		-	-			$\widehat{q_1}$	↓			1	1																													
	-	-																																							
	$\widehat{q_1}$	↓																																							
	1	1																																							
3 – À gauche Nb d'états : 1 Alphabet : {0;1;-} Description : Il faut faire tout le contraire du précédent.	<table><tr><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>$\widehat{q_1}$</td><td></td></tr><tr><td></td><td>↓</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>-</td><td>-</td><td>-</td><td>-</td><td></td></tr></table>		1	1	1	1						$\widehat{q_1}$			↓						-	-	-	-																	
	1	1	1	1																																					
				$\widehat{q_1}$																																					
	↓																																								
	-	-	-	-																																					
4 – Tout allumer Nb d'états : 1 Alphabet : {0;1;-} Description : Cette fois, il faut deux instructions.	<table><tr><td></td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td></td></tr><tr><td></td><td>$\widehat{q_1}$</td><td></td><td></td><td></td><td>↓</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>		0	1	0	0	1	0	1			$\widehat{q_1}$				↓						1	1	1	1	1	1	1													
	0	1	0	0	1	0	1																																		
	$\widehat{q_1}$				↓																																				
		1	1	1	1	1	1	1																																	
5 – Alternateur Nb d'états : 2 Alphabet : {0;1;-} Description : Il faut utiliser deux états.	<table><tr><td></td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>$\widehat{q_1}$</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>↓</td><td></td></tr><tr><td></td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td></td></tr></table>		-	-	-	-	-	-	-									$\widehat{q_1}$									↓			0	1	0	1	0	1	0					
	-	-	-	-	-	-	-																																		
							$\widehat{q_1}$																																		
							↓																																		
	0	1	0	1	0	1	0																																		
6 – Séquenceur Nb d'états : 4 Alphabet : {0;1;-} Description : Il faut utiliser 4 états.	<table><tr><td></td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td></td></tr><tr><td></td><td>$\widehat{q_1}$</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>↓</td></tr><tr><td></td><td>0</td><td>-</td><td>1</td><td>-</td><td>0</td><td>-</td><td>1</td><td>-</td><td>0</td></tr></table>		-	-	-	-	-	-	-	-			$\widehat{q_1}$																		↓		0	-	1	-	0	-	1	-	0
	-	-	-	-	-	-	-	-																																	
	$\widehat{q_1}$																																								
									↓																																
	0	-	1	-	0	-	1	-	0																																