Calculabilité, décidabilité.

Qu'est-ce que veut dire calculer ? Un ordinateur peut-il tout calculer ?

C'est en ..... que ..... (1912-1954) a apporté des réponses à ces questions.

FIGURE 1 – .....

1 Programme en tant que donnée

Nous allons tout d'abord expliciter un point important qui sera le fondement de la théorie de la calculabilité : un programme est aussi une donnée. Cela peut paraître étonnant à première vue puisqu'on est habitué à traiter :

— les programmes dans des fonctions
— les données dans des variables

Fonctions et variables sont des objets de nature différente en apparence. Si on se raccroche à ce que l'on connaît en python, une fonction se déclare avec le mot clé ..... et une variable s'initialise avec l'opérateur d'affectation. .....

Prenons en exemple l'algorithme qui retourne le carré d'un nombre **x** passé en paramètre. On peut l'écrire à l'aide d'une fonction Python :

```
1  def carre(x):
2          return ...
```

Dans ce programme Python, **carre** est une fonction et **x** est une donnée. Ils ne semblent pas être de nature comparable :

```
1  >>> carre(7)
2  49
```

Et pourtant, à y regarder de plus près, notre algorithme programmé dans la fonction **carre** n'est rien d'autre qu'une succession de caractères. On peut même pousser la réflexion jusqu'à créer une chaîne de caractère contenant ce programme :

```
1  mon_programme = "def carre(x):\n\ return x*x"
```

Maintenant l'algorithme est devenu une variable. On peut alors construire une machine universelle capable d'évaluer n'importe quelle donnée contenant un algorithme formalisé dans le langage Python :

```
1  def universel(algo, *args):
2      exec(algo)
3      ligne1 = algo.split('\n')[0]
4      nom = ligne1.split('(')[0][4:]
5      return eval(nom+str(args))
```

A présent je peux appeler ma machine universelle en lui passant en données :

— la variable contenant mon algorithme
— les arguments sur lequel celui-ci va travailler et obtenir la réponse

```
1  >>> universel(mon_programme, 7))
2  49
```

Dans l'exemple ci-dessus, vous pouvez constater que le programme et les données sur lesquelles il agit sont de même nature : ce sont 2 variables passées en paramètres à ma fonction universelle.

Conclusion : Un programme est aussi une .....

Exercice n° 1

Connaissez-vous d'autres situations où un programme est considéré comme une donnée ?

2 Calculabilité

La calculabilité est une branche de l'algorithmique qui s'intéresse aux questions suivantes :
- Peut-on tout calculer à l'aide d'un ordinateur ?
- Que signifie calculer à l'aide d'un ordinateur ?
- Peut-on concevoir un algorithme permettant de savoir si un programme passé en paramètre
  - va se terminer ?
  - va provoquer une erreur ?
  - est correct et ne contient pas de bugs ?

Ce sont des questions fondamentales au cœur de l'algorithmique et de l'informatique en général.

Illustration du théorème fondamental de la calculabilité

Définition : Une fonction **f** avec un argument **x** est calculable s'il existe un programme (algorithme) pour calculer **f(x)**.

Combien existe t-il de programmes (d'algorithmes) ?

L'ensemble des programmes est ..... puisqu'on peut l'assimiler à une une suite finie de caractères.

Combien existe t-il de fonctions ? L'ensemble des fonctions est un ensemble ..... (cela peut être démontré)

Shéma :

Conclusion :

Théorème 2.1. - *Fondamental de la calculabilité* -
Les fonctions non calculables c'est à dire non programmables sont (infiniment) plus nombreuses que celles qui le sont.

Remarque : Un infini dénombrable est un infini que l'on peut compter : $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ sont dénombrables par opposition à $\mathbb{R}$ qui est indénombrable. Il est impossible de numéroter les nombres réels. L'infini indénombrable des nombres réels est bien plus grand que l'infini dénombrable des entiers. De même que la majorité des nombres sont des nombres réels (bien plus nombreux que les entiers).

A retenir : La majorité des fonctions ne sont pas calculables autrement dit la plus grande partie des "problèmes" que l'on peut définir en mathématiques n'ont pas de solution algorithmique.

Conclusion : Peut-on tout programmer ?.....

3 Le problème de l'arrêt

Le premier problème explicite non calculable a été décrit par Turing en 1936 : Il s'agit du problème de l'arrêt qui s'énonce ainsi : étant donné un algorithme **A** prenant un paramètre **m** , existe-t-il un algorithme permettant de décider si **A(m)** s'arrête ?

C'est ce que l'on appelle un problème de .....

3.1 La conjecture de Syracuse

Un exemple de cette indécision est la conjecture de Syracuse, encore non résolue. Voici son énoncé : Soit un entier n . On définit la suite u_n par récurrence ainsi :

— si n est pair, $u_{n+1} = n/2$

— si n est impair, $u_{n+1} = 3n + 1$

u_0 étant donné, on peut ainsi calculer les termes de proche en proche.

Par exemple si on part de 7, on obtient la suite suivante :

.....

A ce jour, tous les nombres essayés conduisent inévitablement à ce cycle 4, 2, 1 mais nul n'a été capable de le démontrer.

Exercice n° 2 conjecture de Collatz ou conjecture de Syracuse

Ecrire une fonction **syracuse** prenant en paramètre un nombre entier strictement positif et renvoyant la liste des termes de la suite jusqu'à ce qu'elle atteigne 1.

3.2 Le problème des nombres de Lychrel

On choisit un entier N par exemple 1059.

Tant que N n'est pas un palindrome.

On additionne N avec son miroir (9501 est le miroir de 1059).

N prend la valeur de la somme calculée ci-dessus.

Exercice n° 3

1. Poursuivre l'algorithme ...

Exercice n° 4 Lichrel

1. Écrire une fonction **lichrel** prenant en paramètre un nombre entier et renvoyant la liste des sommes obtenues jusqu'à l'obtention d'un palindrome.
2. Tester avec 1059, puis 89, puis 196

3.3 Preuve de Turing

S'il existait une solution au problème de l'arrêt, alors la conjecture de Syracuse ou le problème des nombres de Lychrel serait résolue car on saurait prédire l'arrêt de l'algorithme.

Pour prouver que le problème de l'arrêt n'est pas calculable, Turing en 1936 a fait un raisonnement par ..... On suppose que le problème de l'arrêt est calculable c'est à dire qu'il existe un programme **arret** permettant de décider si un algorithme s'arrête. Ce programme aurait cette forme en Python :

```
1 def arret(A, m):
2         ... # Tout est ici !
3         if ...:
4                 return True
5         else:
6                 return False
```

Le vrai problème est bien sûr de compléter les ... mais on suppose ici que quelqu'un à su le faire. On peut alors écrire la fonction **paradoxe** ci-dessous :

```
def paradoxe():
        if arret(paradoxe): #paradoxe est une donnee
                while True:
                        print("Bravo !")
        else:
                return True
```

Si paradoxe s'arrête, ...

Si paradoxe ne s'arrête pas, ...

Conclusion : .....

Cette contradiction montre donc que le problème de l'arrêt est .....

Exercice n° 5 Proof in english

Visionner cette preuve danscette vidéo (en anglais, facile à comprendre)

Exercice n° 6 Défis

Démontrer qu'il est impossible de écrire un algorithme **div0** qui détecte les divisions par 0 dans un calcul.

4 Conclusion

La détection de bugs (comme des boucles infinies par exemple) est une activité cruciale en informatique. Il existe des programmes capables de détecter des erreurs dans d'autres programmes : les environnements de développement modernes comme VScode ou PyCharm sont capables de souligner vos erreurs en python alors même que vous tapez le programme, et cela constitue un grand gain de temps pour le développeur.
En France, depuis l'accident du vol 501 d'Ariane 5 en 1996 dû à une erreur de programmation, le développement de programmes de preuves de correction a connu une forte croissance.
Un bel exemple de progrès réalisé grâce à ces programmes est le logiciel de la ligne de métro automatique 14 (Météor) à Paris : cette ligne a été mise en service sans test en condition réelles : son programme a été mathématiquement prouvé sans fautes. Depuis sa mise en service, aucun bugs n'a été à déplorer.
Malheureusement, un algorithme général permettant de prouver qu'un programme fonctionne n'existe pas, cela fait partie des très nombreux problèmes indécidables. Il n'y aura donc jamais de système permettant de s'assurer que n'importe quel programme est fiable, même si c'est réalisable pour quelques exemples particuliers comme le Météor.

5 Bilan et suite

Une fonction non calculable est donc une fonction pour laquelle aucun programme n'existe. Elles sont "beaucoup plus nombreuses" que les fonctions calculables. En voici qqes exemples :

- Problème de l'arrêt
- Égalité de deux programmes
- Un programme capable de dire si un autre programme est correct !
- TP sur le professeur Z (Suite de votre travail... voir bfourlegnie.com)