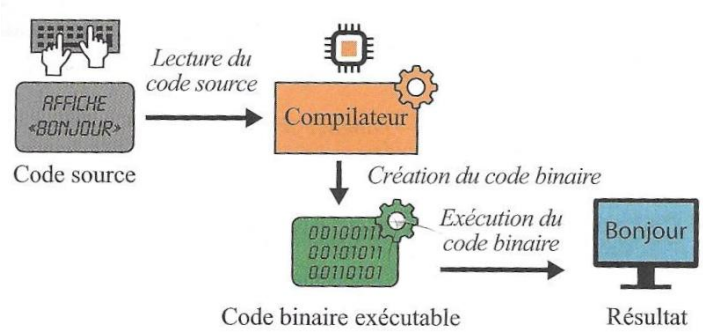
Séquence 5 – Programme en tant que donnée, calculabilité et décidabilité .
Terminale NSI – 2020/2021

1) Programme en tant que donnée.

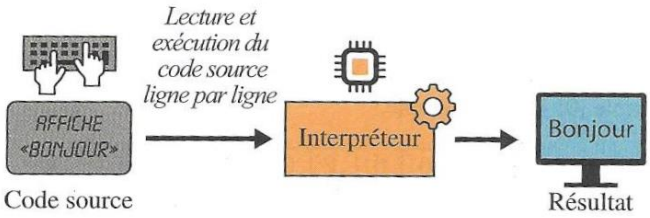
Le principe de fonctions en tant que données que nous avons étudié dans l'activité peut facilement se généraliser à la notion de programme prenant en entrée et/ou fournissant en sortie un autre programme, comme par exemple les systèmes d'exploitation qui sont des programmes manipulant d'autres programmes.

Pour illustrer ce principe, il est intéressant d'étudier les mécanismes d'interprétation et d'exécution de programmes par la machine, en fonction des types de langages employés. En effet, le processeur ne comprend que le langage machine, qui est une suite d'opérations codées en binaire difficilement intelligibles par un être humain. Ainsi, les langages de plus haut niveau qui nous permettent de réaliser des programmes intelligibles seront traduit en binaire de différentes manières :

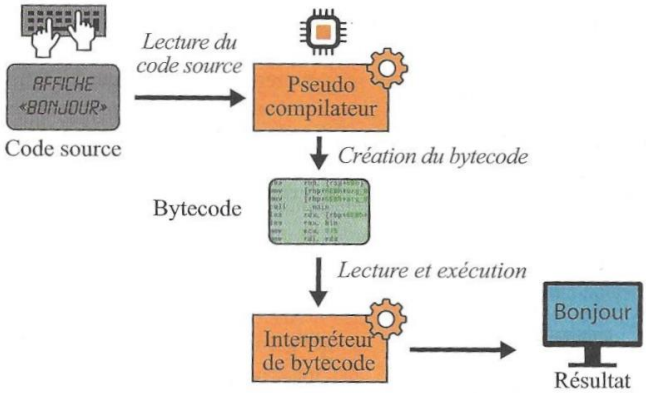
- Les **langages compilés** (ex : C, C++, Pascal, Caml, …) dans lequel le code source écrit par le programmeur est transformé en code binaire par un logiciel appelé **compilateur**.



- Les **langages interprétés** (ex : javascript, Node JS, PHP, …), dont le code source est utilisé en entrée d'un logiciel appelé **interpréteur**, qui l'exécute ligne par ligne en décidant à chaque étape ce qu'il va faire ensuite.



- Les langages **semi-interprétés** (ou mixtes), comme le langage Java, requièrent une compilation différente de celles vue précédemment, car elle ne transforme pas le code source en instructions binaires.
  Elle se contente de préparer le travail de l'interpréteur en réécrivant les instructions du code source dans un langage intermédiaire (bytecode), qui sera plus facile à traiter par l'interpréteur, appelé traducteur dynamique, ou encore compilateur « à la volée » - *JIT (just in time) compiler* en anglais.
  C'est un peu le cas de Python.



2) Calculabilité.

Alan Turing

La notion de calculabilité est apparue en 1936 avec Alan Turing (1912-1954), avec la présentation de sa « machine de Turing ». Cette machine conceptuelle, considérée par certain comme précurseur de l'architecture de Von Neumann (1946), et donc de l'ordinateur moderne, n'a pas de finalité pratique, bien qu'elle ait été simulée de diverses manières. Elle permet plutôt d'expliciter de manière précise ce qu'est un programme, de présenter de manière rigoureuse des algorithmes de tous genres et de trouver les limites à ce qui est calculable. C'est l'objet de la thèse (qui n'est pas un théorème, car non démontrée) de Church-Turing stipulant :

« Si un problème est tel qu'il existe un algorithme le résolvant, alors il existe une machine de Turing qui résout le problème ».

Cette thèse mène alors à la notion de calculabilité.

Définition : Un problème est calculable s'il existe une machine de Turing le résolvant.

L'étude de la machine de Turing, dont la définition est relativement rudimentaire, n'est pas un objectif pour cette année, et il n'est pas gênant de remplacer « machine de Turing » par « programme Python » dans cet énoncé. Si vous souhaitez découvrir la machine de Turing :

https://www.youtube.com/watch?v=X610pII4_J8

3) Décidabilité.

David Hilbert

Le mathématicien David Hilbert (1862-1943) présenta une liste de 23 problèmes mathématiques non résolus en 1900, auxquels il ajoutera le **problème de la décision** en 1928, qui pose la question suivante :

« Y a-t-il un algorithme qui, étant donné un système formel et une proposition logique dans ce système, pourra décider sans ambiguïté si cette proposition est vraie ou fausse dans ce système formel ? »

Définition : Un problème est décidable si et seulement si, il existe un algorithme qui le résout.

Cela sous-entend que l'algorithme se termine en un nombre fini d'opérations.

Exemple : Voici un problème : un entier donné est-il un nombre premier ? Ce problème est-il décidable ?

```python
from math import sqrt

def estPremier(n:int)->bool:
    assert isinstance(n,int) and n>0, "n est un enitier naturel non nul"
    if n==1 :
        return False
    else :
        for i in range(2,int(sqrt(n))):
            if n%i==0 :
                return False
        return True
```

Il existe ainsi un algorithme de test de primalité, donc ce problème est décidable. (il nous resterait cependant à en prouver sa correction…)

Alan Turing a démontré en 1936 que tous les problèmes n'étaient pas décidables à l'aide d'un contre-exemple, mettant en jeu notamment la notion de programme ou de fonction en tant que donnée : le **problème de l'arrêt** qui est le suivant :

« Existe-t-il un programme qui, prenant en entrée un autre programme et son entrée, détermine si ce programme finit par s'arrêter avec cette entrée ou boucle indéfiniment ? »

4) Preuve du théorème de l'arrêt.

Alonzo Church

A la question précédente, Turing répond par la négative en le prouvant grâce à une classe de machine de Turing particulière, la « machine de Turing universelle ». Indépendamment, Alonzo Church (1903-1995) propose une preuve s'appuyant sur son système formel du λ-calcul.

L'étude de ces principes, n'étant pas non plus un objectif cette année, nous allons présenter ici un preuve « moderne » par l'absurde en nous appuyant sur un programme Python.

Posons ainsi comme **hypothèse de départ** qu'il existe une fonction Pyhton $arret(prog, x)$ qui termine toujours son execution et renvoie $True$ si la fonction Python $prog$ s'arrête pour un argument x.

Nous pouvons alors construire une fonction $diag(x)$ de la manière suivante :

```
1  def diag(entree):
2      if arret(entree,entree):
3          while True :
4              pass
5      else :
6          return True
```

Quel est le résultat de l'appel de $diag(diag)$?

- Si $diag(diag)$ s'arrête, alors $arret(diag, diag)$ renvoie vrai, et on entre dans une boucle infinie, donc $diag(diag)$ ne s'arrête pas, ce qui est contradictoire.

- Si $diag(diag)$ ne s'arrête jamais, alors $arret(diag, diag)$ renvoie $False$, et $True$ est renvoyé en résultat, et $diag(diag)$ s'arrête. Là encore, nous sommes face à une contradiction.

Ainsi, tous les cas nous mène à des résultats absurdes, notre hypothèse de départ sur l'existence d'un tel programme est donc fausse. Le problème de l'arrêt est donc indécidable !