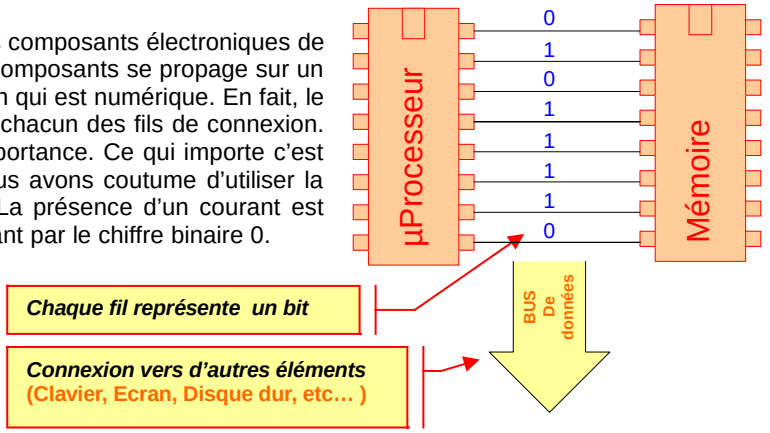
Langage de bas niveau (langage machine)

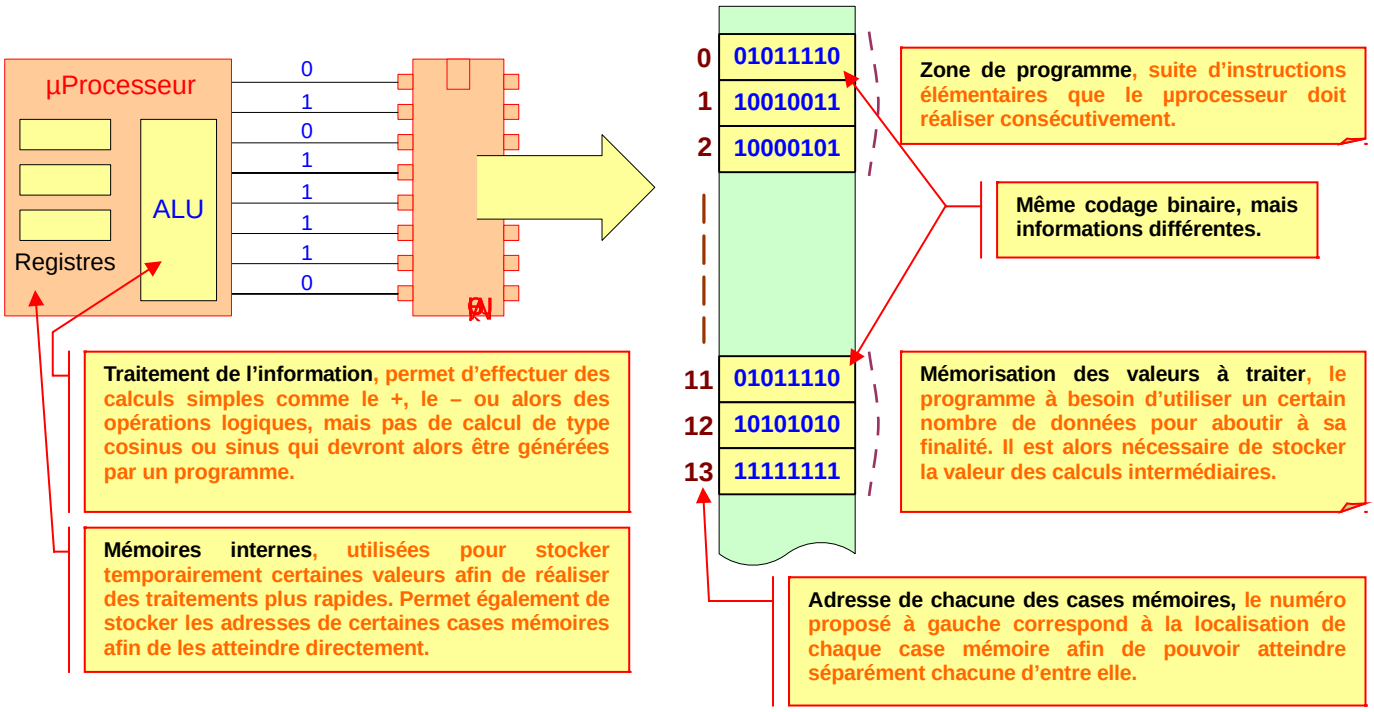
Pourquoi le code binaire ?

Les composants à l'intérieur de l'ordinateur sont des composants électroniques de type numérique. L'information véhiculée entre les composants se propage sur un ensemble de fils de connexion, et c'est cette information qui est numérique. En fait, le système contrôle la présence ou pas d'un courant sur chacun des fils de connexion. La valeur même de l'intensité du courant n'a pas d'importance. Ce qui importe c'est de savoir uniquement si le courant passe ou pas. Nous avons coutume d'utiliser la numération binaire pour représenter cet état de fait. La présence d'un courant est symbolisée par le chiffre binaire 1 et l'absence du courant par le chiffre binaire 0.



Organisation interne :

Si la communication entre les différents composants s'effectue en binaire, cela veut également dire qu'à l'intérieur de tous les composants, tout est structuré de façon binaire. Le microprocesseur s'occupe de réaliser tout un ensemble de traitements élémentaires qui sera déterminé par un programme, et cet ensemble d'instructions élémentaires devra être écrit et stocké dans la mémoire centrale de l'ordinateur. Ce programme est en fait une suite d'instructions écrites en binaire dont chacun des codes correspond à une action spécifique du microprocesseur, comme par exemple, l'opération +. Toutefois, pour que cette simple opération puisse être réalisée, il est nécessaire d'avoir au moins deux opérandes. Donc, cela veut dire que des données doivent être stockées dans un autre endroit de la mémoire centrale de l'ordinateur. Ces données sont également exprimées de façon binaire.



Orientation des langages :

Le premier langage informatique est donc le langage binaire. Seulement, il est très fastidieux d'écrire directement en binaire, surtout qu'il existe un très grand risque de se tromper. Un 1 peut très vite se transformer en 0, ce qui peut avoir des conséquences désastreuses. Pour éviter ces erreurs, l'idée a été d'utiliser le codage hexadécimal qui est en fait une autre façon de représenter le binaire (L'hexadécimal est une concaténation de quatre chiffres binaires). Finalement, que l'on parle du binaire ou de l'hexadécimal, il s'agit de la même chose, c'est-à-dire du langage machine.

Ce langage machine pose un autre problème ; le code correspondant à une instruction d'un processeur ne fonctionne pas nécessairement pour un autre processeur. En effet, chaque processeur possède son propre codage binaire. Du coup, il n'est pas possible de déployer des applications sur des plate-formes où les systèmes d'exploitations sont différents (surtout si les machines ne possèdent pas les mêmes microprocesseurs).

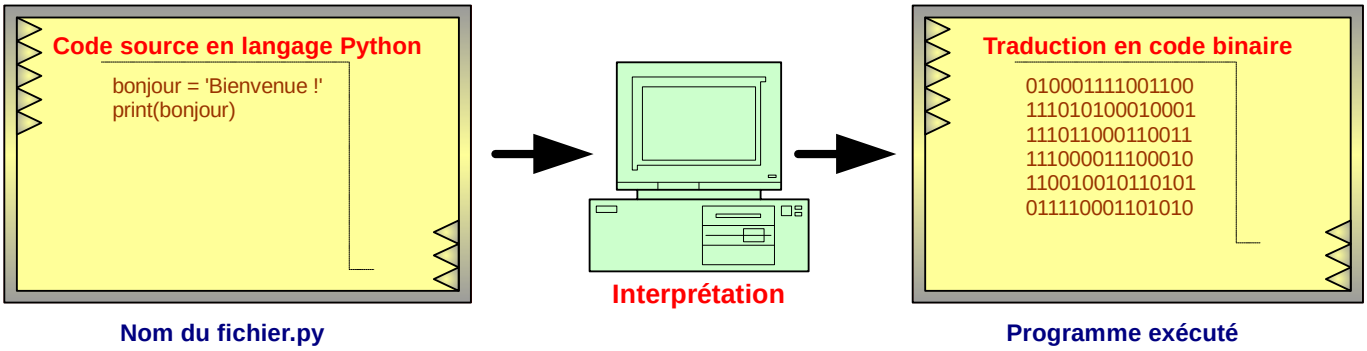
> *L'idée a été donc de proposer des langages de plus haut niveau qui soient d'une part plus proche de la pensée humaine, et qui soient également non tributaire du processeur utilisé.*

Langage de haut niveau - compilation et interprétation

Le langage Python est un langage qui s'approche fortement de la pensée humaine et donc plus facilement compréhensible par le programmeur. C'est à la fois un langage structuré mais aussi Orienté Objet. Malgré tout, le seul langage compréhensible par le microprocesseur reste le langage machine. Il faut donc qu'il existe un système qui analyse notre code en langage évolué (fichier source avec l'extension **py**) et qui le transforme en un code écrit en langage machine.

Il existe deux solutions pour cela. Soit nous effectuons une traduction définitive avec la production d'un fichier binaire qui sera nécessaire pour exécuter le programme. Ce système s'appelle un compilateur et joue donc le rôle de traducteur. C'est ce qu'utilise par exemple le langage C++.

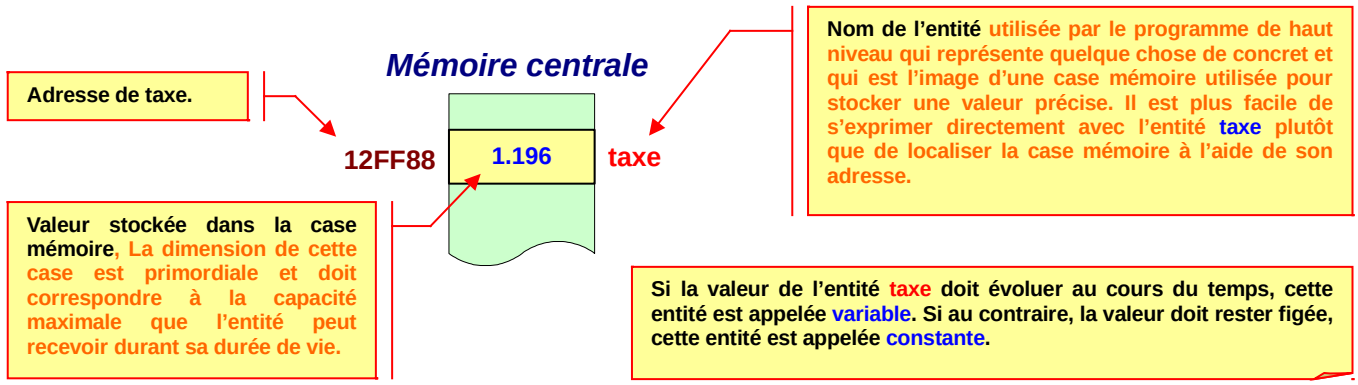
*Soit nous effectuons une traduction « à la volée » sans génération de fichier supplémentaire, ce système s'appelle un interpréteur. C'est ce dernier qui est utilisé par le langage python. L'avantage, c'est que nous n'avons pas besoin de générer un fichier binaire, l'inconvénient toutefois c'est que la traduction sera effectuée à chaque fois que nous lancerons le programme défini dans le code source. Le temps d'exécution est alors plus conséquent. Lorsque nous utilisons un interpréteur, le code source est appelé « **script** ».*



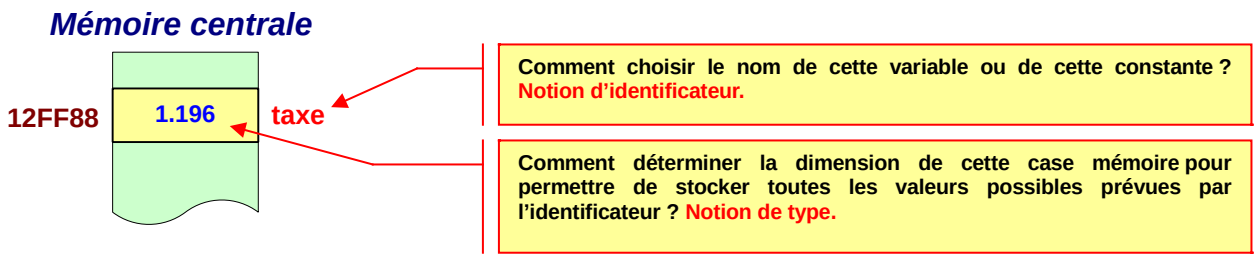
Remarque : Pour tous les langages de programmation compilés ou interprétés, le **code source** est toujours un texte. Donc, pour écrire votre programme, un simple éditeur de texte suffit. Toutefois, il sera nécessaire de posséder un compilateur ou un interpréteur pour réaliser la traduction vers le code binaire (**code objet**).

Mémorisation d'une donnée dans un langage de haut niveau

Un programme dans un langage de haut niveau correspond à une suite d'instructions. Le programme à besoin d'utiliser un certain nombre de données pour aboutir à sa finalité. Il est alors nécessaire de stocker la valeur des calculs intermédiaires.



Questions par rapport à la structure d'une case mémoire :



Définition d'une instruction et notion d'identificateur

La plus petite unité indépendante d'un programme Python est l'instruction. En langage naturel, cela correspond à une phrase. L'instruction est constituée ***d'identificateurs*** de ***symboles***, ainsi que des ***mots réservés***.

Quelques exemples :

```
âge = 37
âge += 3
mineur = âge < 18
âge_actuel = 'Âge = ' + str(âge)
```

Les identificateurs servent à donner des noms aux variables, aux fonctions, aux objets, etc. C'est le programmeur qui choisi le nom de l'identificateur, ce qui lui offre une grande souplesse. Toutefois, il existe un certain nombre de limites.

Les identificateurs peuvent contenir des lettres minuscules ou majuscules, des chiffres, ou le caractère spécial de soulignement « _ ». Par contre, ils ne doivent pas commencer par un chiffre, mais il peuvent posséder des lettres accentuées. Pour finir, les espaces ne sont pas admis dans l'identificateur, sinon, cela voudrait dire qu'il en existe plusieurs.

Le langage Python fait une différence entre les minuscules et les majuscules. Attention, c'est généralement une source d'erreur. Par convention, on utilise très souvent les minuscules pour les variables.

Exemples d'identificateurs :	Salut	carre_entier	_9	var3
Identificateurs non valables :	Hello!	Premier?	7val	9_
Identificateurs différents :	carré	Carré		

Types de données

Un langage informatique manipule de l'information, comme le terme l'indique. Celle-ci est stockée dans les cases mémoire de l'ordinateur sous forme de bits. Cependant, il est rare que l'on est besoin de manipuler ces bits en tant que tels. En général, on souhaite plutôt utiliser des entités plus sophistiquées, comme des entiers, des réels, des booléens, etc. Chacune de ces entités va elle-même être codée sur un certain nombre de bits dans la mémoire.

> *Un langage évolué, comme le Python, permet d'utiliser des données de haut niveau, comme des nombres entiers par exemple, en se chargeant lui-même de la « basse besogne » consistant à convertir les bits de mémoire en ce type de donnée, ou inversement (lors de l'interprétation).*

Une donnée possède un type qui indique deux choses importantes :
- l'ensemble des valeurs dont elle fait partie,
- l'ensemble des propriétés qui la caractérise

Par exemple le type entier « ***int*** » (abréviation de l'anglais *integer*), qui est le type très utilisé dans les langages de haut niveau a généralement pour ensemble de valeurs tous les nombres de -2.147.483.648 à 2.147.483.647 compris (pour microprocesseur compatible 32 bits). En réalité, Python peut largement dépasser ces capacités si besoin et avoir une infinité de chiffres, suivant les limites de la mémoire (notion de variable dynamique qui s'autoadapte). Par contre, il n'est pas possible de prendre, par exemple, la valeur réelle « 1,5 » pour ce type de données puisqu'elle ne correspond pas à la notion d'entier. Cette façon de voir fait penser tout à fait, à l'ensemble de définition que l'on utilise en mathématique.

Parmi les propriétés qui le caractérisent, on trouve un grand nombre d'opérations possibles, comme l'addition « **+** », la soustraction « **-** », la multiplication « ***** », la division entière « **/** », le modulo « **%** », etc.

Le type réel « ***float »*** possède un ensemble de valeurs différent de celles du type ***« int »,*** et certaines opérations comme la division modulo « % » n'ont aucun sens sur ce type. Les propriétés du type ***« float »*** sont donc différentes de celles du type « ***int ».*** Nous peuvons dire également que comme pour les mathématiques, la division entière n'a rien à voir avec une division de nombres réels.

Déclaration d'une variable associée à son type

Une donnée est une brique élémentaire dans un programme que l'on caractérise par son ***type***, d'une part, et par sa valeur actuelle d'autre part. Nous venons de voir ce qu'est un type; la valeur actuelle de la donnée (par exemple 12 pour un entier) est sujette à modification en général, sous la contrainte qu'elle reste dans l'ensemble de valeurs du type. Par contre, le type de la donnée reste toujours le même; en conséquence, les propriétés d'une donnée, qui sont celles de son type, sont constantes.

> *La déclaration d'une variable en Python est assez particulière par rapport à d'autres langages de programmation puisque vous n'avez pas de mot réservé associé au type, c'est la valeur initiale (constante littérale) qui détermine le type. D'ailleurs, pour qu'une variable soit définie, elle doit être impérativement initialisée (justement pour connaître et évaluer son type).*

Vous avez dans la page suivante tous les types disponibles dans le langage Python avec des exemples associés. Il existe également des foncions prédéfinies dans le langage Python (reconnaissable par les parenthèses au niveau de l'argument) qui sont très utiles, notamment, la fonction **type()** qui permet de déterminer le type d'une variable, et la fonction **print()** qui permet d'afficher son contenu.

Type	Premiers programmes en Python

Les nombres entiers	`>>> entier = 18 # déclaration d'une variable entière`
	`>>> entier = entier + 15 # changement de valeur dans la variable`
	`>>> print(entier) # affichage du contenu de la variable`
	`33`
	`>>> type(entier) # connaissance du type de la variable`
	`<class 'int'>  # pour les très grands nombres, il existe aussi le type long`

Les nombre réels	`>>> réel = 3.5 # déclaration d'une variable réelle`
	`>>> réel += 12 # changement de valeur dans la variable avec une constante entière`
	`>>> print(réel) # affichage du contenu de la variable`
	`15.5`
	`>>> type(réel) # connaissance du type de la variable`
	`<class 'float'>`

Les booléen	`>>> test = False # déclaration d'une variable booléenne`
	`>>> test = réel < 20 # test avec la variable réelle précédente`
	`>>> print(test) # affichage du contenu de la variable`
	`True`
	`>>> type(test) # connaissance du type de la variable`
	`<class 'bool'>`

Les chaîne de caractères	`>>> bonjour = "Aujourd'hui, il fait beau."  # déclaration d'une chaîne`
	`>>> message = 'Que dis-tu ? "Salut".'  # déclaration d'une autre chaîne`
	`>>> print(bonjour, message)`
	`Aujourd'hui, il fait beau. Que dis-tu ? "Salut".`
	`>>> type(message)  # connaissance du type de la variable`
	`<class 'str'>`

Les listes	`>>> premiers = [1, 3, 5, 7, 11, 13, 17] # déclaration d'une liste d'entiers`
	`>>> print(premiers)  # affichage du contenu de la liste`
	`[1, 3, 5, 7, 11, 13, 17]`
	`>>> for premier in premiers: print(premier, end=" ")`
	`1 3 5 7 11 13 17  # affichage successif de chaque élément de la liste`
	`>>> type(premier) # connaissance du type de chaque élément de la liste`
	`<class 'int'>`
	`>>> type(premiers) # connaissance du type, ici donc la liste`
	`<class 'list'>`

Les tuples	`>>> paires = (0, 2, 4, 8, 10, 12, 14) # déclaration d'un tuple de nombres paires`
	`>>> print(paires)  # affichage du contenu du tuple`
	`(0, 2, 4, 8, 10, 12, 14)`
	`>>> for nombre in paires : print(nombre, end=' ')`
	`0 2 4 8 10 12 14  # affichage successif de chaque élément du tuple`
	`>>> type(nombre)  # connaissance du type de chaque élément du tuple`
	`<class 'int'>`
	`>>> type(paires)  # connaissance du type, ici donc le tuple`
	`<class 'tuple'>`

Les dictionnaires	`>>> répertoire = {}  # Création d'un dictionnaire vide`
	`>>> répertoire['manu'] = '05-89-99-65-18'  # Ajout d'un nouvel élément`
	`>>> répertoire['alice'] = '03-02-88-12-34' # Ajout d'un autre élément`
	`>>> répertoire['bruno'] = '06-09-08-25-77' # Ajout d'un autre élément`
	`>>> print(répertoire)  # Affichage complet du répertoire`
	`{'bruno': '06-09-08-25-77', 'manu': '05-89-99-65-18', 'alice': '03-02-88-12-34'}`
	`>>> print(répertoire['manu'])  # Affichage d'un numéro de téléphone`
	`05-89-99-65-18`
	`>>> type(répertoire)  # Connaissance du type, ici un dictionnaire`
	`<class 'dict'>`

Les ensembles	`>>> entiers = {5, 6, 15}  # déclaration d'un ensemble déjà rempli`
	`>>> vide = set()  # déclaration d'un ensemble vide`
	`>>> entiers.add(18)  # ajout d'un nouvel élément dans l'ensemble`
	`>>> entiers.add(6)  # l'ajout ne se fait pas, il existe déjà`
	`>>> print('entiers =', entiers, ': vide =', vide)`
	`entiers = {18, 5, 6, 15} : vide = set()`
	`>>> print(5 in entiers)`
	`True`
	`>>> type(entiers)`
	`<class 'set'>`