Les variables

1. Usage

Une variable sert à représenter une valeur par un nom. Vous retrouvez ici un concept assez proche de ce que vous connaissez en mathématiques ou en physique. Cependant, il y a quelques différences :

- En informatique, une variable est toujours connue, car elle est associée à une case mémoire qui contient sa valeur. Le concept d'inconnue n'existe pas !
- En informatique, une variable évolue, sa valeur change au cours du temps suivant les actions effectuées durant l'exécution du programme. Lorsque l'on modifie une variable, la nouvelle valeur écrase la précédente et l'ancienne valeur est oubliée. En mathématiques et en physique, les inconnues ont une unique valeur !

2. Nommage

Si vous deviez retenir une seule consigne de cet ouvrage, ce serait sûrement celle-ci car elle est la plus importante dans la vie d'un informaticien. Nous insistons car elle sera souvent oubliée, maltraitée, contournée par habitude et cela conduira à l'échec. Nous voulons parler du choix du nom d'une variable. Nos cousins les mathématiciens ou les physiciens ont pour usage de choisir des noms d'inconnues les plus courts possible. Ainsi, on voit fleurir sur les pages de nos livres et de nos cahiers des x, des y ou des α et nous y sommes habitués. Pourquoi ce choix ? Pour deux raisons principales : d'une part, ces matières se sont historiquement construites à l'époque du crayon/papier et le premier objectif dans une page de calcul était de limiter la quantité d'informations à écrire. D'autre part, les exercices les plus compliqués ne nécessitant que quelques inconnues, les alphabets grec et latin semblent suffisants pour couvrir les besoins. Ainsi, on peut même avoir des lettres associées à un concept : par usage R désigne une résistance, I une intensité, d une dimension et x une abscisse… Le terrain reste bien balisé.

Cependant, l'homo informaticus a connu une autre histoire. Il utilise le clavier et n'est plus limité par la taille de la page papier. Une "petite application" fait dans les 1 000 à 10 000 lignes de code : une calculette, une horloge, un post-it de bureau. Une application moyenne compte environ 100 000 lignes : un éditeur de texte ou un éditeur d'image basique. Dès que l'on pense à des applications lourdes : un éditeur de texte complet (Word, Office), un navigateur internet, un client e-mail, un logiciel graphique (Photoshop, Illustrator), un site web professionnel, on peut compter plusieurs millions de lignes de code. La priorité devient tout autre : lisibilité et organisation. Ce sont les deux maîtres mots de l'informaticien. Il doit à tout prix structurer sa pensée, organiser son code et le rendre le plus lisible possible. Prenons ces deux lignes de code écrites avec des instructions identiques, mais dans deux styles différents :

```
v = a * (1+t)
```

ou

```
PrixTTC =  PrixHT * (1+TVA)
```

Aucun commentaire n'est nécessaire. Vous serez sûrement tenté par le premier choix par facilité ou par habitude. Cependant, une fois que vous aurez écrit une dizaine de lignes dans le même style, la lecture va vite devenir rock'n'roll et ce sera l'accident. Il faut être conscient qu'il y a beaucoup de choses à faire en programmant : mettre en place une structure logique censée effectuer un traitement pas forcément évident à comprendre, gérer la syntaxe du langage, taper au clavier sans se tromper, lire le sujet, gérer le bruit environnant… Bref, tout cela occupe déjà 90 % de notre esprit. En choisissant de prendre des noms de variables anonymes, voire confus, voire ambigus, on ajoute une couche de complexité qui finit par embrouiller notre pensée. Combien de fois trouvera-t-on une variable `t` qui désigne un prix ? Logique ! Car il y avait déjà une variable `p` utilisée pour désigner le nombre d'articles dans le Caddie ! Une autre fois, on choisira `p1`, `p2` et `p3` comme variables pour désigner les prix de chaque article et `p4` pour désigner la somme totale… La confusion est proche, vous finirez par croire que vous avez acheté quatre

articles, et lorsque l'on vous demandera la somme totale, vous écrirez p1+p2+p3+p4 et l'accident se produira.

Il semble difficile de trouver des noms simples qui désignent les variables. Lorsque le programme est court, c'est-à-dire moins de 10 lignes, que l'énoncé pose clairement les données à utiliser, la tendance chez le débutant est de persister dans le premier style, car les habitudes sont bien ancrées et la prise de risque reste finalement faible sur si peu de lignes. Cependant, cela reste une pratique dangereuse qui nuit à la mise en place de l'algorithme. Ce style d'écriture devient plus délicat lorsqu'on commence à dépasser 100 lignes de code.

Généralement, le problème n'est pas si important car les débutants créent un code sur une échelle de temps très courte : une heure, un jour, maximum une semaine. En général, on se souvient de ce que représente chaque variable et le nommage devient une tâche accessoire. Au bout du temps imparti, le programme est oublié et passe à la trappe. Cependant, dès que vous entrez sur des productions plus longues ou plus complexes, il suffit qu'une semaine de vacances s'immisce au milieu de votre production, et au retour, c'est le grand flou. Cette remarque semble évidente pour les développeurs qui travaillent en équipe. Lorsque vous reprenez le code d'un partenaire qui n'a fait aucun effort de nommage, la lecture du code devient vite un sac de nœuds et les modifications deviennent alors très difficiles à mener.

Il ne faut pas sous-estimer les pièges apportés par un certain laisser-aller sur le nommage des variables. Il n'est pas nécessaire de choisir des noms très longs ou des quasi-phrases pour désigner une variable, cela finirait par nuire à la lisibilité et ce serait contre-productif. Mais déjà, vous pouvez faire commencer un nom de variable désignant un prix par P ou nommer un vecteur entre deux points A et B par vAB, c'est un bon début ! Nommer judicieusement les variables reste une tâche difficile, mais l'effort en vaut la chandelle.

3. Créer des variables !

Il semble aussi difficile, pour les débutants comme pour les programmeurs avancés, de créer une variable pour chaque chose à calculer. Par exemple, il reste fréquent de trouver deux variables pour stocker trois valeurs. Supposons que nous ayons deux variables et que nous calculions leur somme. Au lieu de créer une troisième variable intitulée somme, vous préférez stocker le résultat dans la première variable. Premièrement, vous écrasez la valeur qu'elle contenait, pensant que vous ne vous en servirez plus.

Ensuite, vous stockez une somme dans une variable dont le nom ne correspond pas à une somme, c'est dangereux. Troisièmement, lorsque vous aurez finalement besoin de la valeur écrasée, il sera impossible d'y avoir accès directement. Il faudra alors effectuer un tour de passe-passe pour la recalculer en faisant une soustraction. Tout cela est épuisant et dangereux.


Il y a sûrement beaucoup de ressources à économiser pour "sauver" la planète, mais limiter le nombre de vos variables n'a aucun intérêt pour l'écosystème ! Créez des variables chaque fois que cela est nécessaire, c'est-à-dire pour chaque résultat de calcul normalement.

Voici un autre exemple :

```
(1) t = 18
(2) t = t * (1+k)
(3) p = p + t
(4) Afficher( t / (1+k) , t , p)
```

Voici la version lisible avec un nommage clair :

```
(1) Prix = 18  # prix hors taxe
```

```
(2) Prix = Prix * (1+TVA)
(3) Somme = Somme + Prix
(4) Afficher( Prix, Prix/(1+TVA), Somme)
```

Dans ce code, le prix hors taxe défini à la ligne 1 est écrasé à la ligne 2 pour être remplacé par le prix TTC. La valeur 18 n'existe plus et lorsque le programmeur veut l'afficher à la ligne 4, il doit faire le calcul en divisant par la grandeur (1+TVA). Voici la version utilisant trois variables, pas vraiment plus longue, mais où tout coule de manière limpide entre chaque étape :

```
PrixHT = 18
PrixTTC = PrixHT*(1+TVA)
Somme = Somme + PrixTTC
Afficher( PrixHT, PrixTTC, Somme)
```

Il semble aussi difficile de créer des variables intermédiaire même lorsqu'elles sont nécessaires. Le débutant préfère en général un copié-collé. Par exemple, vous voulez dessiner deux cercles de rayon 100, le premier à la position (x1,y1) et le second à la position (x1+200,y1), puis un segment qui joint ces deux centres.

Vous êtes tenté d'écrire :

```
DessineCercle(x1,y1,100)
DessineCercle(x1+200,y1,100)
DessineSegment(x1,y1,x1+200,y1)
```

Cela est juste et cela fonctionne, ce sont les deux arguments massue que vous êtes tenté d'utiliser pour vous convaincre que tout va bien. Mais dans ce code, les dangers sont multiples. D'une part, s'il y a une erreur de frappe cachée dans ces trois lignes, nous vous souhaitons déjà bien du courage pour la trouver. D'autre part, un code est destiné à évoluer au cours d'un projet. Si finalement, on vous demande par la suite d'éloigner les deux centres d'une distance de 250, vous allez changer la première occurence de la valeur 200 en 250 et vous risquez fort d'oublier de changer la deuxième occurrence présente à la troisième ligne. Heureusement, lors de l'affichage graphique, vous verrez tout de suite votre oubli. Cependant, dans un contexte autre, sans affichage, le problème sera invisible et vous créerez un bogue à cause de l'oubli d'un report de modification sur l'ensemble de vos copiés-collés. Nous vous proposons une autre écriture où le centre du deuxième cercle dispose de ses propres variables. Ainsi, le tracé du segment se positionne sur les deux centres en utilisant les variables créées. Si vous modifiez la position d'un centre, le segment se replace automatiquement au bon endroit. Ce n'était pas le cas dans le premier exemple.

```
Rayon = 100
DessinerCercle(x1,y1,Rayon)
x2 = x1 + 250
y2 = y1
DessinerCercle(x2,y2,Rayon)
DessineSegment(x1,y1,x2,y2)
```

Voici un critère d'évaluation de la qualité de votre code : dans ce dernier exemple, en moins de 2 secondes, quasiment en un coup d'œil, on peut comprendre ce que font ces lignes. Dans la version précédente, il vous faut au moins 10 secondes pour reconstruire dans votre esprit le dessin avec les deux cercles et vous apercevoir que le segment joint leurs centres. Dans cette deuxième version, la compréhension est quasi immédiate, car le segment est positionné exactement là où les centres des cercles se trouvent.

Pourquoi avoir choisi de rajouter une variable `Rayon` ? Est-ce purement parce que nous sommes en train de parler de l'importance de créer des variables ? Là aussi, il faut idéalement créer cette variable pour des raisons de lisibilité et pour éviter certains problèmes. Dans cette deuxième version, nous voyons immédiatement que les deux cercles utilisent la même variable et par conséquent nous comprenons qu'ils ont le même rayon. Dans la première version, il fallait se concentrer pour s'en apercevoir et nous ne savions pas si c'était le fruit du hasard ou une contrainte forte. Créer des variables fait ainsi partie du travail de documentation du code.

4. Affectation

Il s'agit de l'opération la plus fréquente dans un programme. Elle consiste à associer un nom à un objet. Dans le langage Python, un objet désigne un nombre, une chaîne de caractères, mais aussi des choses plus complexes comme une date, une liste de nombres ou encore des images… En Python, il n'y a aucune différence de traitement entre ces différents types d'objets. Par souci de simplicité, nous présentons uniquement des exemples avec des nombres, mais tout ce que nous énonçons ici s'applique à toutes les autres catégories d'objets.

Le terme « objet » prête à confusion, car dans les langages découlant historiquement du langage C (C++/Java/C#/PHP/VBA...), il y a une distinction forte entre les nombres et les objets. Ils obéissent à des règles différentes et ils sont gérés de manière différente dans le code. Ainsi, dans ces langages, le terme "objet" s'oppose aux types de base dont font partie les nombres entiers ou les nombres à virgule. Dans le langage Python, cette opposition n'existe pas.

L'affection est une opération se lisant de droite à gauche : on prend une valeur à droite du symbole = pour l'associer à la variable à gauche de ce symbole. Ainsi, dans la partie gauche d'une affectation, on doit uniquement trouver le nom d'une variable. Dans la partie droite, on peut avoir :

- **Une valeur** : comme le nombre 12 ou la chaîne de caractères "Bonjour".
- **Une variable** : on va donc lire la valeur à laquelle cette variable est associée.
- **Une expression** : une succession de calculs fournissant un résultat, par exemple `abs(a+3)`.

Une expression se compose de valeurs, de variables, de fonctions (valeur absolue, racine carrée…) ou d'opérations (+,-,/…). Il n'y a pas de limite à la longueur d'une expression, elle pourrait faire plusieurs pages !

Une variable a un sens différent suivant qu'elle se trouve à gauche ou à droite du symbole =. Si la variable se trouve à gauche, elle désigne la variable qui sera associée au résultat de l'expression à droite du symbole. Si la variable se trouve à droite, elle sera automatiquement remplacée par la valeur à laquelle elle est associée. Ainsi, en écrivant `a = b + 5`, on remplace de suite la variable `b` par la valeur à laquelle elle est associée, par exemple 7. Ensuite, on évalue l'expression en effectuant le calcul, puis on associe la valeur 12 à la variable `a`. À aucun moment, l'ancienne valeur associée à la variable `a` n'est utilisée.

À tort, ou à raison, le symbole = a été choisi pour représenter l'affection. Mais il est inadéquat, car en mathématiques on peut écrire aussi bien a=b ou b=a. Dans l'ensemble des langages informatiques, le symbole = se lit de droite à gauche. On évalue d'abord l'expression à droite et on l'associe au nom de variable se trouvant à gauche. Le symbole flèche gauche ← aurait été plus judicieux. Il est d'ailleurs présent dans les langages de certaines calculatrices scientifiques.

Quelles sont les actions effectuées lors de l'exécution de cette ligne ?

```
prix_ticket = 10 * 5 + 3
```

Au premier abord, il n'y a aucune difficulté et cette question semble la plus simple du monde. Mais, si par curiosité on veut détailler les étapes déroulées par l'interpréteur Python, cela devient un peu plus complexe. C'est une question de culture générale, au jour le jour, elle n'est pas utile. Mais elle permet de comprendre un peu mieux ce qu'il se passe. Voici la réponse :

- **Analyse du nom de la variable prix_ticket** : deux options sont possibles. Soit ce nom est inconnu, et dans ce cas, l'interpréteur crée une nouvelle variable portant ce nom. Soit il est connu, ce qui sous-entend que la variable est déjà associée à quelque chose.
- **Évaluation de l'expression** : à droite du symbole =, on trouve un calcul. L'interpréteur va d'abord évaluer cette expression et trouver la valeur 53 correspondant à un nombre entier.
- **Réservation de la mémoire** : la valeur 53 est stockée dans le processeur, comme elle serait affichée sur votre écran de calculette. Au prochain calcul, elle risque de disparaître, il faut donc lui réserver une place en mémoire. Ce sera dans la mémoire RAM de l'ordinateur. La mémoire RAM est celle qui se compte en gigaoctets : 4 Go, 8 Go, 16 Go. Ne la confondez pas avec le disque dur dont le rôle sert à stocker les fichiers.
- **Stockage** : maintenant qu'une cellule mémoire est réservée, la valeur 51 est écrite à l'intérieur.
- **Association** : une liaison entre la variable **prix_ticket** et la cellule mémoire contenant la valeur 51 est créée.

Les noms de variables doivent respecter une dizaine de règles. Nous donnons les plus importantes ici :

- Le nom ne doit pas commencer par un chiffre !
- Le nom ne doit pas contenir de caractères spéciaux : ! % @ # [+ = \ ( ) [ ] = & : . , ; $..., exception faite du trait souligné appelé underscore "_" présent sur la touche [8] du clavier.
- Le nom peut contenir des caractères accentués du langage courant : é ù ç. Cependant, comme il est difficile de gérer tous les caractères de toutes les langues existantes, on a tendance à utiliser seulement les 26 lettres de l'alphabet, le symbole souligné _ et les chiffres car ils sont présents sur tous les claviers européens.