Introduction à l'algorithmique et à la programmation

DUT Informatique 1ère année, 2018 – 2019

Anthony Labarre
Anthony.Labarre@u-pem.fr

– Cours 1 / 10 –

– Aspects pratiques –

Organisation :

- 10 séances de 2h de cours / TD
- 12 séances de 2h de TP

Evaluation :

- une note de TP : vous rendez chaque TP, tous seront notés
- un projet de programmation
- un examen

Ressources :

- `http://igm.univ-mlv.fr/~alabarre/teaching.php`
- une page sur l'intranet de l'UPEM (voir TPs)

– Plan d'aujourd'hui –

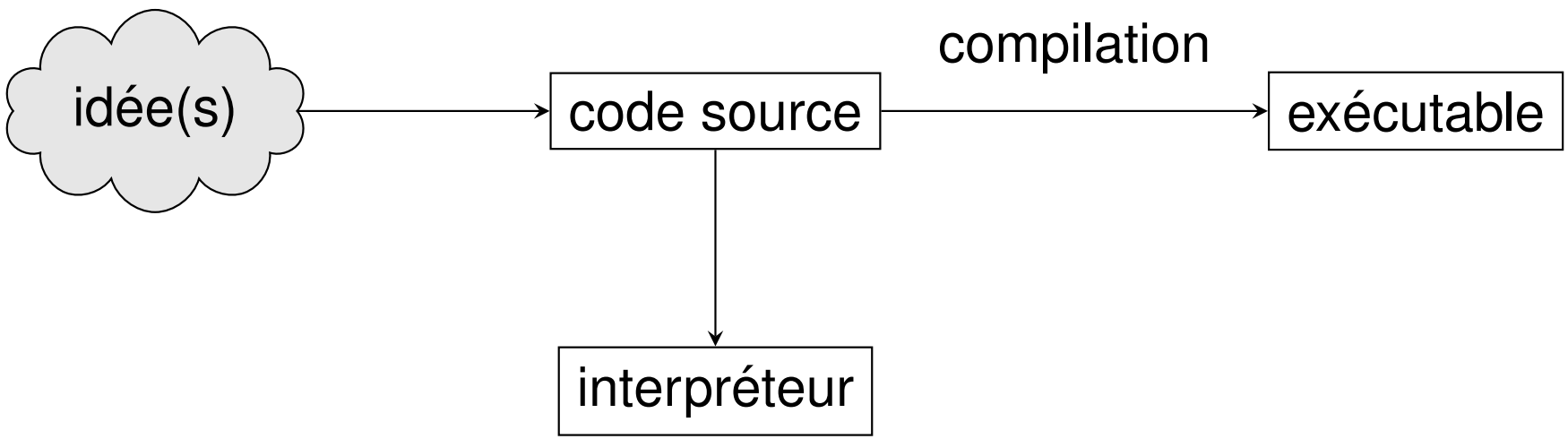
Variables, types et opérations

Instructions et blocs

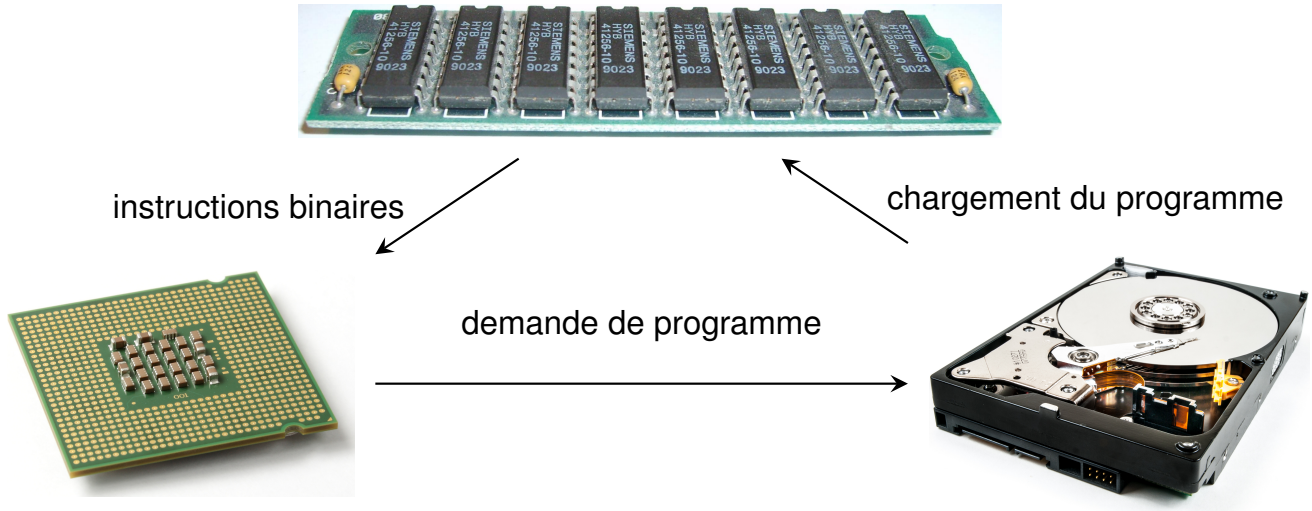
Instruction conditionnelle (if)

– Qu'est-ce que la programmation? –

Programmer : (verbe transitif) "Écrire les programmes informatiques correspondant à l'algorithme de résolution d'un problème" (Larousse), fractionner un problème en instructions codifiées acceptables par la machine.



– Pourquoi un langage? –



- Le processeur ne “comprend” que les instructions binaires;
  - Elles varient d’un processeur à l’autre: certaines changent, d’autres manquent;
  - Programmer en binaire est loin d’être ergonomique;
- Il nous faut donc un **langage de programmation** intermédiaire entre ce qui est naturel pour un humain et ce que le processeur peut comprendre (par exemple: Python, C, C++, Java, ...);

– Introduction sur un exemple –

Un exemple de programme en Python 3 :

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2017 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

C'est un programme un peu avancé. . . mais essayons de comprendre ce qu'il fait.

– Les différents mots –

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2017 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

La **coloration syntaxique** rend le programme plus lisible et varie d'un éditeur à l'autre. Ici, on a :

- en vert et mauve des **mots-clés** ou des **fonctions** du langage Python;
- en rouge du texte;

– Les différents mots –

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2017 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

Les **mots-clés** et **fonctions intégrées** de Python, en vert, sont en anglais:

if = si `input` = entrée `int` = entier **print** = imprimer

Les **opérateurs** de Python (en mauve) aussi:

and = et

– L'indentation –

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2017 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

- Les lignes ne commencent pas toutes au même endroit.
- Ce n'est pas un détail !
- Le positionnement (on dit l'**indentation**) des lignes est important pour leur signification.

Modifier l'indentation modifie le comportement du programme — et peut même l'empêcher de fonctionner.

– La structure –

- Les programmes Python doivent être **structurés** (cf. `modele.py` sur la page du cours):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''Description de ce a quoi sert le module, ses fonctionnalites.'''
# Autres commentaires generaux (auteur, date d'ecriture, adresse e-mail, etc.)

# Imports: inclusion des modules dont vous avez besoin ----------------------
#import os   # exemple, a decommenter en cas de besoin
#import sys  # exemple, a decommenter en cas de besoin


# Fonctions et variables globales -------------------------------------------
def ma_fonction(param1, param2):
    '''Explication de ce que fait la fonction, des resultats renvoyes le cas
    echeant, et de ce a quoi correspondent ses parametres.'''
    pass  # on ne fait rien (a remplacer par ce qu'il vous faut)


# Le corps du programme: c'est la qu'on ecrit le code a executer quand le
# module est passe a l'interpreteur Python
if __name__ == '__main__':
    # Inserez ici les tests de vos fonctions et les autres actions a executer
    pass  # on ne fait rien (a remplacer par ce qu'il vous faut)
```

- On y reviendra; pour l'instant, retenez qu'il faut commencer par `if __name__ == '__main__':`
- Les **commentaires** (`# ...`) expliquent le code;

– En résumé –

- On a vu un exemple avancé illustrant dans les grandes lignes à quoi ressemble un programme;
- On a vu
  - que les lignes sont des instructions à effectuer, dans l'ordre;
  - qu'on peut stocker des valeurs (dans `date`, `age`, ...);
  - qu'on peut effectuer des calculs (`age - 1`);
  - qu'on peut utiliser des fonctions de Python (**`print`**, `input`, `int`);
  - que la structure et l'indentation importent;
  - qu'on peut exécuter le programme `fichier.py` avec la commande `python3 fichier.py`
  - ...
- Dans la suite, on va apprendre pas à pas et dans le détail toutes ces notions (et bien d'autres), afin que vous maîtrisiez les bases de la programmation

Python pas à pas

Variables, types et opérations

– Types de valeurs –

- Les **valeurs de base** possèdent un **type**;
- Le type va notamment déterminer ce qui se passe quand on fait une opération sur des valeurs;

Les principaux types (on en verra d'autres) sont :

Type	**En Python**	**Exemples**
entier	`int`	12, -4, 123545, ...
flottant	`float`	3.14159, -1.5, 12., 4.56e12, ...
booléen	`bool`	`True` (vrai) ou `False` (faux)
indéfini, rien	`None`	`None`
chaîne de caractères	`str`	`'bonjour'`,`'IUT info'`,...

Les majuscules/minuscules sont importantes :

`True` $\neq$ `true`

– Transtypage –

- La fonction `type()` permet de connaître le type d'une valeur;
- On peut aussi demander à Python de changer le type d'une valeur (ce qu'on appelle le **transtypage**):
  - `str(51)` renvoie la chaîne `'51'`;
  - `int()` convertit en entier, quand cela est possible ;
  - `float()` convertit en flottant, quand cela est possible;
  - `bool()` convertit en booléen;

Essayons dans un terminal Python ...

– Quelques exemples –

`int`(4.5) → 4
`int`(`'IUT'`) → **erreur**
`str`(4) → `'4'`
`bool`(4) → `True`

`int`(-4.5) → -4
`float`(4) → 4.
`str`(`True`) → `'True'`
`bool`(0) → `False`

`int`(`'0345'`) → 345
`float`(`'4.5'`) → 4.5
`str`(-4.5) → `'-4.5'`
`bool`(`'IUT'`) → `True`

En pratique, on se sert surtout de :

- `str` qui fonctionne tout le temps
- `int` et `float` appliqués à une chaîne de caractères qui correspond à un nombre
  - Attention: `int`(`'3'`) fonctionne, mais pas `int`(`'3.5'`)!
- `int` appliqué à un `float` pour tronquer les décimales
- `bool` qui donne `False` si son paramètre équivaut à 0, `True` sinon

– Opérations sur les nombres –

- On dispose des opérations usuelles sur les nombres; le type du résultat dépend des types d'entrée:

`+, -, *`	`int`	`float`
`int`	`int`	`float`
`float`	`float`	`float`

- La division (/) donne toujours un `float`
- On dispose également de la division Euclidienne, avec quotient et reste comme en primaire. Le quotient de `x` et `y` est `x // y` et leur reste est `x % y` (**modulo**)
- Il y a enfin l'opération puissance qui se note `x ** y`
- Les opérations suivent les règles de priorités usuelles et on peut utiliser des parenthèses : `(4 + 2) * 1.5`

Quelques exemples ...

– Opérations avec booléens –

- On a les opérations sur les booléens :
  - `and`: le **ET** logique, `x and y` vaut `True` seulement quand `x` et `y` valent `True`
  - `or`: le **OU** logique, `x or y` vaut `False` seulement quand `x` et `y` valent `False`
  - `not`: la **négation** logique, `not(True)` = `False` et `not(False)` = `True`
- Les comparaisons produisent des booléens :
  - Le test d'égalité se fait avec `==`
  - Le test de différence se fait avec `!=`
  - On a aussi `<` `<=` `>` `>=` pour comparer selon l'ordre usuel (ordre du dictionnaire pour les chaînes)

Encore des exemples . . .

– Opérations sur les chaînes de caractères –

- Seuls **+** et * marchent sur les chaînes;
  - Si on utilise **+** sur deux chaînes de caractères, on effectue la **concaténation** des deux chaînes :
    `'IUT' + 'info'` → `'IUTinfo'`
  - "Multiplier" une chaîne par un entier **n** la répète **n** fois :
    `'IUT' * 3` → `'IUTIUTIUT'`

– Autres opérations –

- Il existe beaucoup d'autres opérations sur les chaînes;
- On a accès à plein d'opérations mathématiques (cosinus, ...);
- On verra ça plus tard dans le semestre;

– Variables –

- Une **variable** est un nom qui référence une valeur en mémoire;
- On peut s'en servir dans les calculs;
- Elle a le même type que la valeur qu'elle référence;

– Affectation –

- L'**affectation** d'une variable consiste à lier un nom à une valeur;
- La syntaxe : **nom = expression**, où **expression** est une valeur ou un calcul qui produit une valeur :

```
x = 3
y = 'IUT'
z = x + 2
```

- On peut affecter à nouveau une même variable, et on perd alors le lien avec l'ancienne valeur;

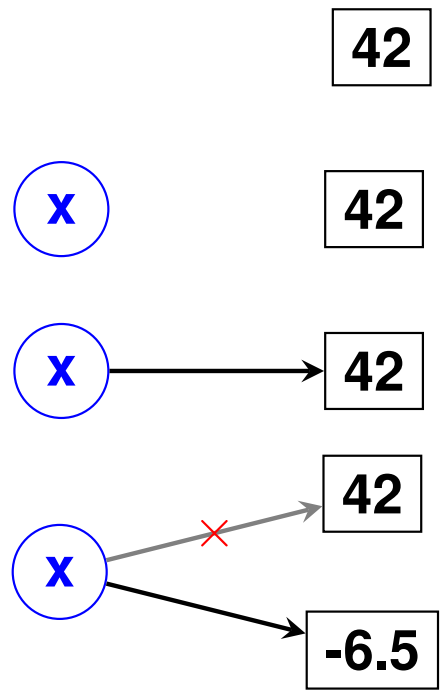
Ce n'est pas du tout le **=** des mathématiques. Il faut le lire comme "prend la valeur" : **x = x + 1**

– Etapes de l'affectation –

```
x = 40 + 2
```

1. On commence par calculer le membre droit, ici on trouve **42**
2. Ensuite on crée le nom pour **x** (sauf s'il a déjà été créé)
3. Enfin on relie la variable à sa valeur

   En cas de réaffectation, le lien d'avant est perdu : `x = -6.5`



– Nommage –

Il y a certaines **règles** de nommage à respecter pour les variables (et les fonctions qu'on verra plus tard):

- le caractère "underscore" `_` est considéré comme une lettre
- on n'utilise jamais d'accent, de cédille, . . . ou d'espace
- les noms commencent par une lettre majuscule ou minuscule, puis sont composés de lettres et de nombres :
  `exemple` `_ex2` `Ex2mpl1` ~~`2016iut`~~
- les mots réservés de Python (voir suite) ainsi que les noms de fonctions prédéfinies sont interdits

– Mots réservés –

Les mots suivants sont réservés pour le langage :

```
and     as      assert    break   class   continue
def     del     elif      else    except  finally
for     from    global    if      import  in
is      lambda  nonlocal  not     or      pass
raise   return  try       while   with    yield
True    False   None
```

Un moyen simple de s'en rendre compte: si votre éditeur affiche le nom de votre variable en gras ou en couleur, c'est qu'il s'agit d'un mot réservé.

Python pas à pas

Instructions et blocs

– Instructions et séquences d'instructions –

```
if __name__ == '__main__':
    date = input(
        "Bonjour! Quelle est votre date "
        "de naissance (jj/mm/aaaa)? "
    )
    jour_naissance = int(date.split("/")[0])
    mois_naissance = int(date.split("/")[1])
    annee_naissance = int(date.split("/")[2])
    age = 2017 - annee_naissance
    if mois_naissance > 9:
        age = age - 1
    if mois_naissance == 9 and jour_naissance > 5:
        age = age - 1
    print("Vous avez", age, "ans")
```

- Comme on a vu dans l'introduction, les instructions sont effectuées dans l'ordre, de haut en bas
- En Python, il n'y a **qu'une instruction par ligne**
- Le flot d'instructions peut être modifié / redirigé par des conditions (**if**), des boucles (plus tard), . . .

– Au passage ... `input` –

```
date = input(
    "Bonjour! Quelle est votre date "
    "de naissance (jj/mm/aaaa)? "
)
```

- La fonction `input(str)` permet à l'utilisateur de saisir une valeur au clavier
- Dans l'exemple ci-dessus:
  1. la chaîne `'Bonjour!  ...'` est affichée à l'écran (comme avec **`print`**);
  2. le programme attend que soit rentrée une chaîne;
  3. le programme affecte cette valeur à la variable `date`;
- La fonction `input` renvoie toujours une chaîne de caractères; on a donc besoin du transtypage pour convertir cette valeur;

Quelques exemples avec `input`

– Blocs d'instructions –

Certaines instructions sont regroupées en **blocs** de la façon suivante :

```
entête du bloc:
    instruction 1 du bloc
    instruction 2 du bloc
    instruction 3 du bloc
instruction hors bloc
```

- L'indentation (le décalage) se fait avec la touche tabulation;
- On peut insérer un bloc dans un bloc, un bloc dans un bloc dans un bloc, ...
- L'indentation fait partie du langage Python, changer l'indentation **change la signification** du programme

Ne mélangez pas tabulations et espaces dans l'indentation!!! Configurez votre éditeur pour qu'il remplace les tabulations par des espaces.

Python pas à pas

Instruction conditionnelle (if)

– La conditionnelle : le `if` –

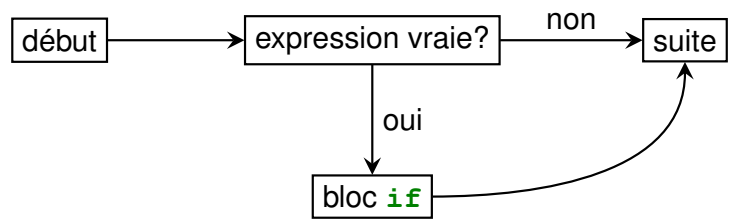
```
if mois_naissance > 9:
    age = age - 1
if mois_naissance == 9 and jour_naissance > 5:
    age = age - 1
print("Vous avez", age, "ans")
```

- Dans cet exemple, on teste si `mois_naissance > 9`;
  - Si c'est le cas, on effectue le bloc qui suit, où `age` chute d'une unité
- On teste ensuite si `mois_naissance == 9` ET si `jour_naissance > 5`;
  - Si c'est le cas, on retire une unité à `age`
- Dans tous les cas, on termine par afficher l'âge calculé;

– La conditionnelle : le `if` –

La forme la plus simple est

```
# début
if expression:
    # instruction 1 du if
    # instruction 2 du if
    # ...
# suite
```



- **expression** est une expression qui renvoie un booléen, qui est donc évaluée à `True` ou `False`
- les instructions du bloc du **`if`** sont effectuées **uniquement si** l'expression est évaluée à `True`
- dans tous les cas, le programme reprend à **l'instruction après `if`**

– Conditions dans un `if` –

- On peut mettre autant de conditions que l'on veut dans un `if`; par exemple:

```
if a < b and (b == 2 * c or b == 3*c):
    # faire quelque chose
```

- Ces conditions sont évaluées **dans l'ordre où elles sont écrites**;
- Si la première condition d'un `and` est fausse, alors la suite n'est **pas évaluée**;

Attention: Python 2 $\neq$ Python 3

Les deux versions **ne sont pas** compatibles !

– Quelques liens utiles –

Indispensable

- Télécharger Python: `http://www.python.org/`

Editeurs

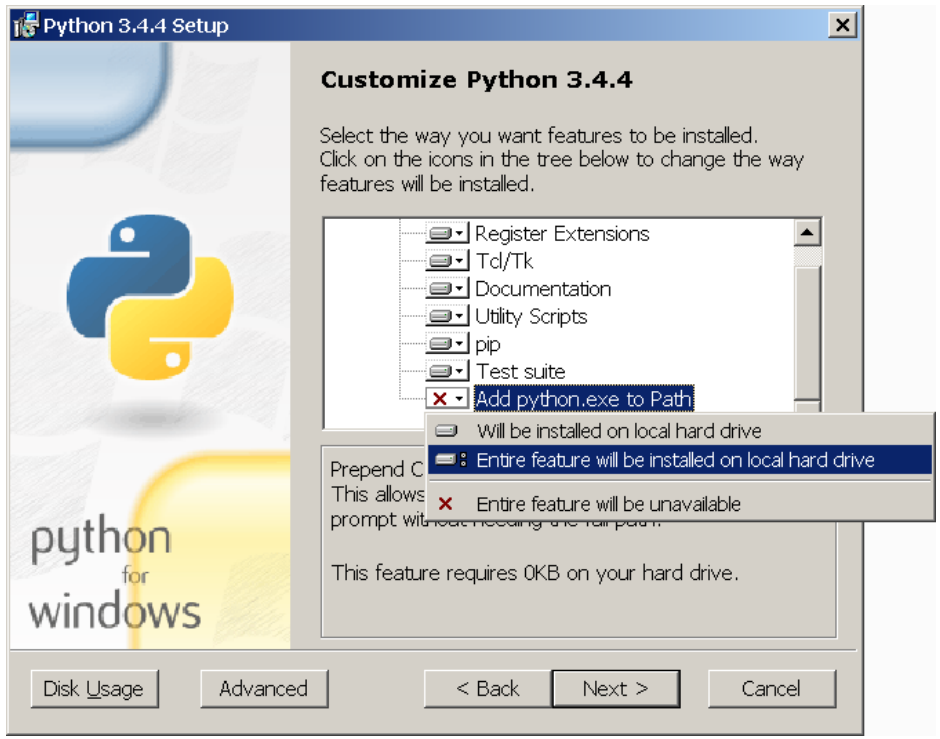
- Geany: `https://www.geany.org/`
- PyCharm: `https://www.jetbrains.com/pycharm/`
- Notepad++: `https://notepad-plus-plus.org/`
- Kate: `http://kate-editor.org/`
- SublimeText: `https://www.sublimetext.com/`
- ...

Autres

- Visualiser Python: `http://pythontutor.com/`

– Installation de Python –

- Sous Ubuntu / Debian: `sudo aptitude install python3` ou `sudo apt-get install python3`
- Sous Windows: suivez le programme d'installation **mais ajoutez python à la variable PATH!**



Sans cela, vous ne pourrez pas taper `python` dans le terminal;