

Algorithmes et Pensée Computationnelle

Programmation de base

Le but de cette séance est d'aborder des notions de base en programmation. Au terme de cette séance, l'étudiant sera capable de :

- Comprendre les différentes représentations de nombres.
- Définir des variables et comprendre les types de variables existants.
- Utiliser des notions d'algèbre booléenne.
- Ecrire des branchements conditionnels.

Les langages qui seront utilisés pour cette séance sont Java et Python. Assurez-vous d'avoir bien installé IntelliJ. Si vous rencontrez des difficultés, n'hésitez pas à vous référer au guide suivant : [tutoriel d'installation des outils et prise en main de l'environnement de travail](#).

1 Représentation de nombres entiers

Question 1: (🕒 5 minutes) Entiers non signés

Sur 8 bits, convertir $113_{(10)}$ en base binaire.

💡 Conseil

Faire un tableau comme présenté dans la diapositive 10 du cours (programmation de base). Essayer de décomposer le nombre en une somme de puissances de 2.

>_ Solution

Méthode 1 : Utiliser la division par 2 comme vu dans la première séance d'exercices

Méthode 2 : Utiliser les puissances de 2 pour décomposer le nombre (vu également dans la première séance d'exercices).

$$113 = 64 + 32 + 16 + 1 = 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 1 * 2^0$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	1	1	0	0	0	1

On obtient donc $01110001_{(2)}$

Question 2: (🕒 5 minutes) Entiers signés négatifs

En utilisant le résultat de la question précédente, convertir sur 8 bits $-113_{(10)}$ en base binaire.

💡 Conseil

- Il faut prendre la représentation sur 7 bits d'un nombre entier non signé.
- On rajoute un 8ème bit qui sera le signe.
- Pour cet exercice, reprendre l'expression non signée de la question précédente (question 1 - Entiers non signés) et changer le premier bit en conséquence.
- Le premier bit vaut 0 pour un nombre positif et 1 pour un nombre négatif.

>_ Solution

$$113_{(10)} = 01110001_{(2)}$$

En changeant le premier bit à 1 pour faire passer le nombre en nombre négatif on obtient :

$$-113_{(10)} = 11110001_{(2)}$$

Question 3: (🕒 5 minutes) Complément à 1

Écrire le complément à 1 de $-113_{(10)}$.

Quelle est la différence entre cette méthode et la précédente ?

💡 Conseil

- L'opposé de 0 en binaire est 1 et inversement.
- Pour étudier la différence, il faut regarder les différentes manières d'exprimer -0 en binaire (se référer à la diapositive 11 du cours de la semaine 3).

>_ Solution

$$113_{(10)} = 01110001_{(2)}$$

0	1	1	1	0	0	0	1
not	not	not	not	not	not	not	not
1	0	0	0	1	1	1	0

avec cette méthode, $-113_{(10)} = 10001110_{(2)}$

L'intervalle de cette méthode ne change pas par rapport à la précédente. Par contre, l'expression de -0 sera différente.

Signé : $-0_{(10)} = 10000000_{(2)}$

Complément à 1 : $-0_{(10)} = 11111111_{(2)}$

Les deux ont le même intervalle : $[-127_{(10)}, +127_{(10)}]$

Question 4: (🕒 5 minutes) Complément à 2

Quel est le complément à 2 de $-113_{(10)}$ et quelle est l'utilité de cette représentation ?

💡 Conseil

Exemple du cours avec $87_{(10)}$

$$87_{(10)} = 01010111_{(2)}$$

a	0	1	0	1	0	1	1	1
b	1	0	1	0	1	0	0	0
c	1	0	1	0	1	0	0	1

a : convertir le nombre en binaire

b : inverser tous les bits

c : rajouter 1 au nombre pour obtenir le complément à 2

On obtient donc $-87_{(10)} = 10101001_{(2)}$

>_ Solution

Ici, il suffit donc d'ajouter 1 au complément à 1 de $-113_{(10)}$

On obtient donc $10001111_{(2)}$

Concernant l'intervalle, il est changé, étant donné qu'il n'existe plus qu'une seule représentation possible pour $-0_{(10)}$.

Le nouvel intervalle est donc $[-128_{(10)}, +127_{(10)}]$

Question 5: (🕒 10 minutes) Floating point

Voici la représentation en binaire d'un nombre à virgule flottante :

signe	exposant	mantisse
0	10110101	010000010000000000000001

Que vaut cette représentation en base 10? Utiliser la représentation des floating points (avec un biais de 127). Arrondir les résultats intermédiaires et la valeur finale au 3ème chiffre significatif après la virgule.

💡 Conseil

- Se référer à la diapositive 15 du cours (programmation de base) pour plus de détails.
- Poser chaque calcul, et ensuite tout fusionner avec la formule.
- L'exposant et la mantisse sont des entiers positifs, donc non signés.
- Pour vérifier votre résultat, vous pouvez utiliser l'outil suivant : <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

>_ Solution

- Signe du nombre : $(-1)^{\text{signe}} = (-1)^0 = 1$
- Exposant en base 10 : $10110101_{(2)} = 181_{(10)}$
- Pour obtenir l'exposant, il faut encore lui appliquer le biais, il faut soustraire 127 à notre résultat. Ici on aura $2^{(181-127)} = 2^{54}$
- Mantisse : $010000010000000000000001_{(2)} = 1 + 1 \cdot 2^{-2} + 1 \cdot 2^{-8} + 1 \cdot 2^{-23} = 1.254_{(10)}$
- Valeur = Signe * Mantisse * $2^{\text{exposant}-127} = 1 \cdot 1.254 \cdot 2^{54} = 2.259 \cdot 10^{16}$

2 Typage

Le but de cette partie des travaux pratiques est de comprendre les notions de typage statique et dynamique, ainsi que leur impact sur l'exécution et la gestion des erreurs.

Question 6: (🕒 5 minutes) Les types de variables

Quelles sont les principaux types de variables (donnez en 3), comment les déclareriez-vous en Python et en Java ?

💡 Conseil

Se référer aux diapositives du cours ou à la documentation officielle des langages de programmation.

>_ Solution

Les principaux types de variables sont les suivants : **Integer** (**int**), **Float**, **String** (**str**), **Boolean** (**bool**).

En Python :

```
1 i = 0
2 f = 3.14
3 s = "Hello World"
4 b = True
```

En Java :

```
1 int i = 0;
2 float f = 3.14f;
3 String s = "Hello World";
4 boolean b = true;
```

Il existe d'autres types de variable, comme par exemple les **double**.

Question 7: (🕒 10 minutes) Typage statique et dynamique

Parmi ces différents programmes, lesquels pourront être exécutés sans problème et lesquels lèveront des erreurs ? Dans le cas où ils génèreraient des erreurs, expliquez la raison de ces erreurs.

Java :

```
1 // Programme 1
2 int variable_1 = 0;
3 int variable_2 = 3;
4 variable_2 = variable_1;
5
6 // Programme 2
7 var variable_1 = 0;
8 var variable_2 = "Hello";
9 variable_2 = variable_1;
10
11 // Programme 3
12 int variable_1 = 0;
13 String variable_2 = "Hello";
14 variable_2 = variable_1;
15
16 // Programme 4
17 var variable_1 = 0;
18 variable_1 = 3.14 ;
19
20 // Programme 5
21 var variable_1 = 0;
22 String variable_2 = "Hello";
23 String variable_3 = "World";
24 variable_2 = variable_3;
```

```

25
26 // Programme 6
27 int variable_1 = 0;
28 variable_1 = 3.14 ;

```

Python :

```

1 # Programme 7
2 variable_1 = 0
3 variable_2 = "Hello"
4 variable_2 = variable_1
5
6 # Programme 8
7 variable_1 = 0
8 variable_1 = 3.14

```

Conseil

En Java, le type est statique, ce qui signifie qu'à partir du moment où vous créez une variable, un type va lui être attribué (par vous ou par le code en lui-même par déduction). Ce type ne pourra pas être changé, et si vous tentez de le faire (en lui attribuant une valeur d'un autre type par exemple), une erreur sera levée.

En Python, le typage est dynamique, il est donc tout à fait possible de changer le type d'une variable sans déclencher d'erreur.

Erreurs fréquentes

Au cas où vous exécuteriez les programmes 2, 4 et 5 et rencontriez l'erreur suivante : **Exception in thread "main" java.lang.UnsupportedClassVersionError: ... has been compiled by a more recent version of the Java Runtime (class file version 54.0), this version of the Java Runtime only recognizes class file versions up to 52.0**, suivez les étapes ci-dessous :

1. Mettez à jour votre JDK en téléchargeant la version adéquate via le lien suivant : <https://www.oracle.com/java/technologies/javase-jdk15-downloads.html>.
2. Une fois votre JDK installé, redémarrez IntelliJ
3. Dans la barre de menus, cliquez sur **Run > Edit Configurations**.
4. Dans la fenêtre qui apparaîtra (capture ci-dessous), sélectionnez la version du JRE la plus récente.

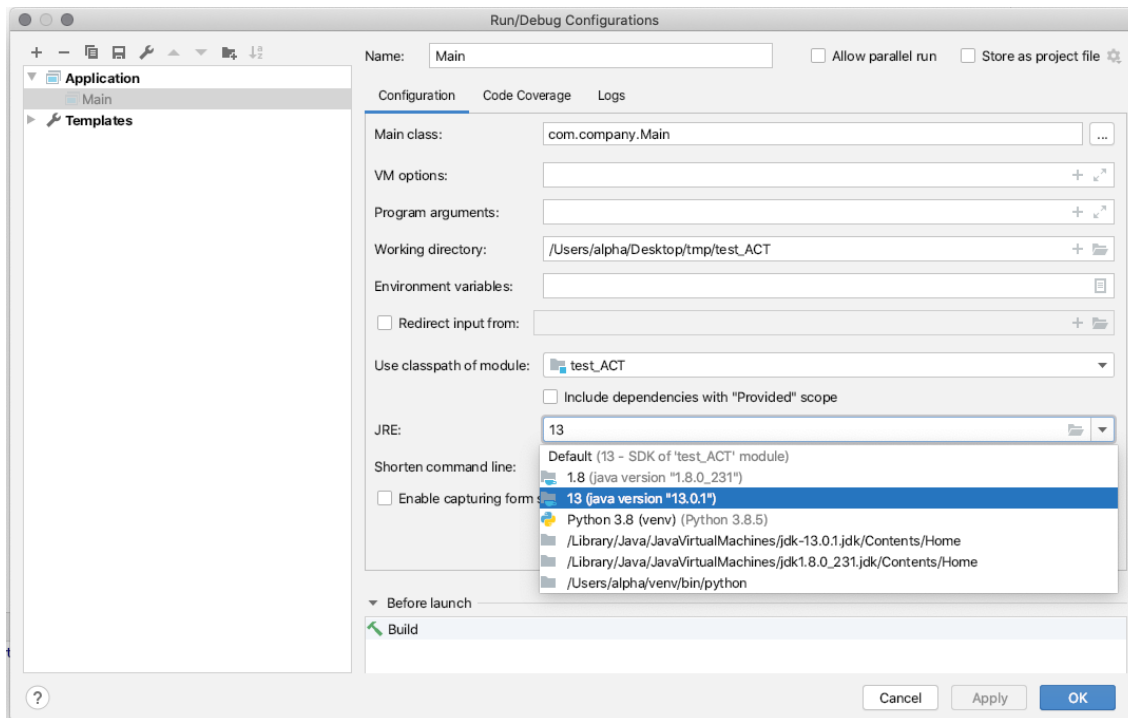


FIGURE 1 – Fenêtre de configuration de l’environnement de développement Java.

>_ Solution

En Java

- Programme 1 : OK
- Programme 2 : Ce code ne fonctionnera pas car on essaye de changer le type de la variable_2, ce qui est impossible avec un typage statique comme en java. Ici le type de la variable est “dédruit” lors de l’exécution du programme.
- Programme 3 : Tout comme le programme précédent, ce code ne fonctionnera pas car on essaye de changer le type de la variable_2, ce qui est impossible avec un typage statique comme en java.
- Programme 4 : Ce code ne fonctionnera pas car on essaye de changer le type de la variable_1, ce qui est impossible avec un typage statique comme en java. Ici le type de la variable est “dédruit” lors de l’exécution du programme.
- Programme 5 : OK
- Programme 6 : Ce code ne fonctionnera pas car on essaye de changer le type de la variable_1, ce qui est impossible avec un typage statique comme en java.

En Python

- Programme 7 : OK
- Programme 8 : OK

3 Opérateurs et conditions Booléennes (Python uniquement)

Le principe d'une valeur booléenne est qu'elle ne puisse contenir que 2 valeurs possibles, soit **True**, soit **False**. Il est possible de les définir en leur associant une de ces valeurs d'emblée ou de les obtenir en effectuant une comparaison. Pour ce faire, il faut utiliser des opérateurs booléens. Voici les plus utilisés : `==` (est égal), `!=` (n'est pas égal), `<` (est strictement plus petit), `<=` (est plus petit ou égal), `>` (est strictement plus grand), `>=` (est plus grand ou égal). Si la condition est satisfaite, on obtiendra **True**, si elle ne l'est pas, on obtiendra **False**. L'utilisation de l'opérateur **not** inversera le résultat.

Dans les exercices suivants, vous devrez anticiper la valeur que la console va vous donner (résultat du print).

Conseil

Utilisez les tables de vérité présentées à la diapositive 7 du cours (Discrete Math Basics).

Question 8: (🕒 5 minutes) Qu'affiche le programme suivant ?

```
1 a = 3
2 b = 2
3 c = 6
4 d = c >= b
5 print(a==b)
6 print(a*b==c)
7 print(d)
8 print(a <= b)
9 print(not d)
```

>_ Solution

```
False
True
True
False
False
```

Question 9: (🕒 5 minutes) Qu'affiche le programme suivant ?

```
1 a = 3
2 b = 2
3 c = 6
4 d = c >= b
5 print(a==b or d)
6 print(a==b and d)
7 print(a==b or a==c or False or d)
8 print(a<c and not (b==2 and not d))
```

>_ Solution

```
True
False
True
True
```

Question 10: (🕒 5 minutes) Qu'affiche le programme suivant ?

```
1 a = True
2 b = False
3 c = True
4 if a or (b and not c) :
5     print("output 1")
6 if b!= c :
7     print("output 2")
8 if a or (not b != (c and a)) :
```

```
9     print("output 3")
10 if not a or ( b==c and not b) :
11     print("output 4")
```

>_ Solution

output 1
output 2
output 3