Algorithmes et Pensée Computationnelle

Programmation de base - Exercices 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 le rôle d'une variable et les types de variables existants.
- Découvrir les différentes représentations de nombres.
- Utiliser des notions d'algèbre booléenne.

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 de la semaine 3. Essayer de décomposer le nombre en une somme de puissances de 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.

Question 3: (*5 minutes*) **Complément à 1**
Ecrire le complément à 1 de $-113_{(10)}$.

Quelle est la différence entre cette méthode et la précédente ?

Conseil

- En programmation l'opposé d'une variable est `not`(la variable). Ici, le même principe s'applique. L'opposé de 0 en binaire est 1.
- 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).

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

Question 5: (*5 minutes*) **Conversion d'un nombre binaire (au format complément à 2) en base 10**

Soit le nombre binaire suivant exprimé sur 8 bits au format complément à 2 : $10010011_{(2)}$. Convertissez ce nombre en base 10.

Conseil

— Utilisez le même tableau que celui de la question 4 des exercices de base (complément à 2).

Question 6: (*10 minutes*) **Floating point**

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

signe	exposant	mantisse
0	10110101	01000001000000000000001

Que vaut cette représentation en base 10 ? Utiliser la représentation des floating point (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 de la semaine 3 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.

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 7: (*5 minutes*) **Les types de variables**

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

> **Conseil**
>
> Se référer aux diapositives du cours ou à la documentation officielle de votre langage de programmation préféré.

Question 8: (*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 apparaitra (capture ci-dessous), sélectionnez la version du JRE la plus récente.

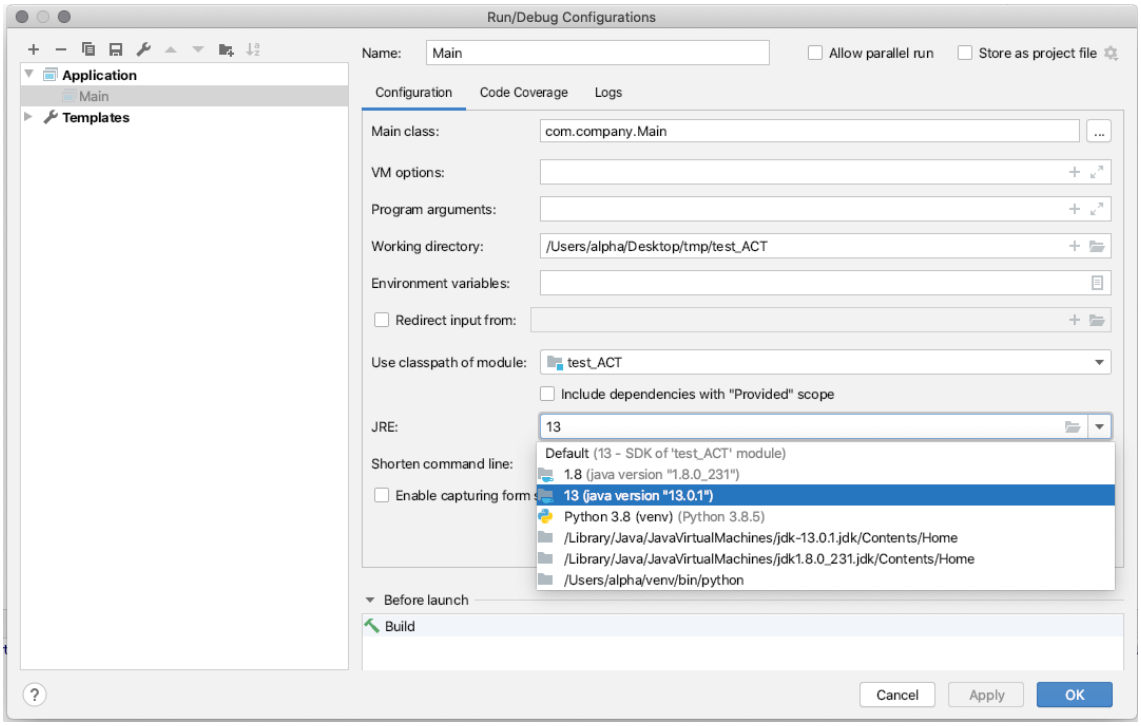

FIGURE 1 – Fenêtre de configuration de l'environnement de développement Java.

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(des) print(s)).

Conseil

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

Question 9: (*5 minutes*) Qu'affichera le programme suivant ?

```
a = 3
b = 2
c = 6
d = c >= b
print(a==b)
print(a*b==c)
print(d)
print(a <= b)
print(not d)
```

Question 10: (*5 minutes*) Qu'affichera le programme suivant ?

```
a = 3
b = 2
c = 6
d = c >= b
print(a==b or d)
print(a==b and d)
print(a==b or a==c or False or d)
print(a<c and not (b ==2 and not d))
```

Question 11: (*5 minutes*) Qu'affichera le programme suivant ?

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