3 — Exercices

ACTIVITÉ

Observer les résultats obtenus par l'expression `5 - 3 - 2` et par l'expression `1 / 2 / 2`.

En déduire la manière dont sont interprétées les soustractions et les divisions enchaînées.

ACTIVITÉ

Réécrire les expressions suivantes en explicitant toutes les parenthèses :

1. `1 + 2 * 3 - 4`
2. `1+2 / 4*3`
3. `1-a+a*a/2-a*a*a/6+a*a*a*a/24`

CORRECTION

1. `(1 + (2*3)) - 4`
2. `1 + ((2/4)*3)`
3. `(((1 - a) + ((a*a)/2)) - ((a*a*a)/6)) + ((a*a*a*a)/24)`

ACTIVITÉ

Réécrire les expressions suivantes en utilisant aussi peu de parenthèses

que possible sans changer le résultat.

1. `1+(2*(3-4))`
2. `(1+2)+((5*3)+4)`
3. `(1-((2-3)+4))+(((5-6)+((7-8)/2)))`

CORRECTION

1. `1 + 2 * (3 - 4)`
2. `1 + 2 + 5*3 + 4`
3. `1 - (2 - 3 + 4) + 5 - 6 + (7-8)/2`

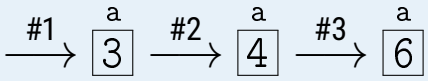
ACTIVITÉ

Déterminer la valeur affichée par l'interprète Python après la séquence d'instructions suivante :

```
a = 3
a = 4
a = a+2
a
```

CORRECTION

Après ces instruction, l'interprète affiche `6`. Voici les états ligne par ligne :



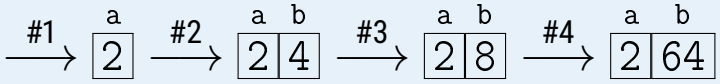
ACTIVITÉ

Déterminer la valeur affichée par l'interprète Python après la séquence d'instructions suivante :

```
a = 2
b = a*a
b = a*b
b = b*b
b
```

CORRECTION

L'interprète affiche la valeur de la variable `b`, soit `64`. Voici les états successifs de l'interprète :



ACTIVITÉ

Dans un notebook, **initialiser** une variable `a` avec la valeur 2, puis **répéter** dix fois l'instruction `a = a * a`.

Observer le résultat. Quelle puissance de 2 a-t-on ainsi calculé ?

Recommencer en affectant cette fois-ci la valeur `2.0` à la variable `a`. **Observer** puis **interpréter** le résultat.

CORRECTION

En procédant ainsi l'interprète a calculé 2^{1024}.

En affectant `2.0` à la variable `a`, on obtient `inf` qui signifie *infini* et indique que le nombre flottant n'est pas représentable car il est trop grand.

ACTIVITÉ

Indiquer ce qu'affichent les instructions suivantes `print("1+")` et `print(1+)`.

CORRECTION

Dans le premier cas la chaîne de caractère `1+` est affichée. Dans le second cas, l'expression (addition entre un nombre entier et rien du tout) comporte une erreur de syntaxe. L'exception `SyntaxError` est levée.

ACTIVITÉ

Indiquer ce qu'il se passe quand on exécute le code suivant :

```
a = input("saisir un nombre : ")
print("le nombre suivant est ", a+1)
```

Rectifier si nécessaire.

CORRECTION

L'expression `a+1` est incorrecte puisqu'elle demande d'effectuer une addition entre `a` qui est une chaîne de caractère et le nombre entier `1`. Cette

opération n'est pas définie en Python.

Pour corriger ce code, il faut par exemple ajouter dans une ligne intermédiaire le code `a = int(a)`.

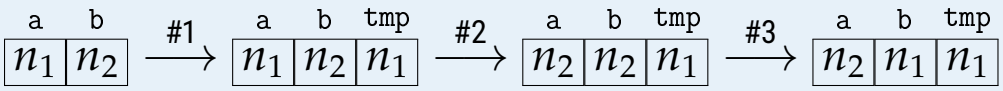
ACTIVITÉ

Indiquer ce que fait la séquence d'instruction suivante en supposant qu'à l'origine les variables `a` et `b` contiennent un nombre entier.

```
tmp = a
a = b
b = tmp
```

CORRECTION

Détaillons les états successifs de l'interprète en supposant que la variable `a` contienne la valeur n_1 et que la variable `b` contienne la valeur n_2.



À la fin de l'instruction, les valeurs enregistrées dans les variables `a` et `b` ont été **permutées**.

ACTIVITÉ

On met deux entiers dans deux variables `a` et `b`, par exemple 55 et 89. On remplace le contenu de `a` par la somme de celui de `a` et de `b`. Puis

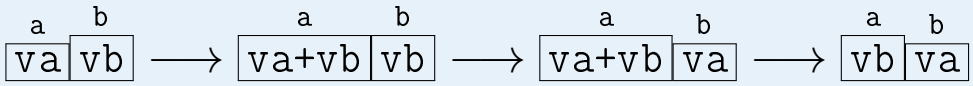
on remplace le contenu de `b` par le contenu de `a` moins le contenu de `b` ? Enfin on remplace le contenu de `a` par son contenu moins celui de `b`.

Que contiennent `a` et `b` à la fin de ces opérations ?

Programme cet algorithme en Python.

CORRECTION

Notons `va` et `vb` les valeurs initiales des variables `a` et `b`.



À la fin de cette séquence d'instructions, les valeurs de `a` et de `b` ont été permutées.

[ ]:
```
a = 55
b = 89

a = a + b
b = a - b
a = a - b
print("a vaut", a, "et b vaut", b)
```

ACTIVITÉ

Écrire un programme qui demande à l'utilisateur les longueurs (entières) des deux côtés d'un rectangle et affiche son aire.

[ ]:
```
texte_l1 = input("Saisir la longueur du premier côté :")
texte_l2 = input("Saisir la longueur du second  côté :")
l1 = int(texte_l1)
l2 = int(texte_l2)
aire = l1 * l2
print("L'aire du rectangle vaut", aire)
```

ACTIVITÉ

Écrire un programme qui demande d'entrer une base (entre 2 et 36) et un nombre dans cette base et qui affiche ce nombre en base 10.

La notation `int(chaine, base)` permet de convertir une `chaîne` représentant un entier dans une `base` donnée en un entier Python.

[ ]:
```
txt_base = input("Saisir la base :")
txt_nb   = input("Saisir le nombre :")
base = int(txt_base)
nb = int(txt_nb, base)
print("le nombre",txt_nb,"écrit en base",
     txt_base,"s'écrit en base 10 :", nb)
```

ACTIVITÉ

Écrire un programme qui demande à l'utilisateur d'entrer un nombre de secondes et qui l'affiche sous la forme d'heures/minutes/secondes.

[ ]:
```
txt_seconde = input("Saisir un nombre de secondes :")
seconde = int(txt_seconde)
minute = seconde // 60
seconde = seconde % 60

heure = minute // 60
minute = minute % 60

print(heure,"h", minute,"min", seconde,"s")
```

ACTIVITÉ

On souhaite écrire un programme qui demande à l'utilisateur un nombre d'œufs et affiche le nombre de boîtes de 6 œufs nécessaires à leur transport. On considère ce programme qui utilise la division euclidienne.

```
n = int(input("combien d'œufs : "))
print(n//6)
```

Tester ce programme sur différentes entrées.

1. Sur quelles valeurs de `n` ce programme est-il correct ?
2. Pourquoi n'est-il pas correct de remplacer `n // 6` par `n // 6 + 1` ?
3. Proposer une solution correcte.

CORRECTION

1. Ce programme n'est correct que pour les nombres `n` multiples de 6.
2. Avec la modification proposée, le programme est correct pour les valeurs qui ne sont pas multiples de 6, mais est devenu incorrect pour les valeurs multiples de 6.
3. Un programme correct est :

```
n = int(input("combien d'œufs : "))
print((n+5) // 6)
```