Notions transversales de programmation

Programmation en Python

Le thème (*Notions transversales de programmation*) correspond au chapitre suivant :

- **Chapitre transversal : Programmation en Python**

Le chapitre transversal est indépendant des autres chapitres. Il peut être traité à tout moment de l'année. Il peut également être enseigné, activité par activité, avec un exercice correspondant. Tous les programmes s'appuient sur des notions d'informatique, en rapport avec le programme de SNT.

A. Le programme

Les capacités exigibles du BO pour ce chapitre sont données ci-dessous.

Capacités attendues du BO traitées dans le chapitre transversal	Contenus	Activités / Exercices
Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.	Affectations, variables, séquences	Activité 1 p. 12 Exercices 1, 2 et 3 p. 23
	Définitions et appels de fonctions	Activités 2 et 6 p. 13 et 17 Exercices 4 et 5 p. 23
	Instructions conditionnelles	Activité 3 p. 14 Exercices 6 à 8 p. 24
	Boucles non bornées	Activité 4 p. 15 Exercices 11 à 14 p. 25
	Boucles bornées	Activité 5 p. 16 Exercices 9 à 14 p. 24-25

B. Description des activités

Activité 1 p. 12 Comment déplacer un robot avec Python ?

Capacité travaillée :

- Affectations, variables, séquences.

Afin de présenter la programmation en langage Python, nous avons choisi de développer une interface ludique. Le but de cette première activité est de se familiariser avec l'interface, en découvrant l'affectation, les variables et les séquences d'instructions en Python.

Les questions 2 et 3 permettent de manipuler les instructions de déplacement de BORDy et de comprendre ce qu'est une séquence d'instructions en langage Python.

Dans la question 4, les élèves doivent faire attention à la syntaxe d'une chaîne de caractères car un code est requis pour ouvrir la porte.

Les questions 5, 6 et 7 permettent d'aborder la notion de variable et d'affectation en langage Python.

À partir de la question 5, les messages lus sont aléatoires afin d'éviter que les élèves récupèrent les codes sans lire les messages, ce qui les oblige à utiliser la variable `message`.

Réponses aux questions :

1. On va à l'adresse indiquée.

2. Un exemple d'instructions Python pour que BORDy sorte de la salle 1 :

```
droite()
droite()
droite()
droite()
droite()
droite()
droite()
droite()
droite()
droite()
droite()
```

3. Un exemple d'instructions Python pour que BORDy sorte de la salle 2 :

```
droite()
droite()
droite()
bas()
bas()
droite()
droite()
droite()
haut()
haut()
haut()
haut()
droite()
droite()
droite()
droite()
bas()
bas()
droite()
```

4. Un exemple d'instructions Python pour que BORDy sorte de la salle 3 :

```
droite()
droite()
droite()
droite()
haut()
haut()
droite()
droite()
droite()
droite()
droite()
droite()
bas()
bas()
deverrouiller('java')
droite()
```

5. Un exemple d'instructions Python pour que BORDy sorte de la salle 4 :

```
droite()
droite()
droite()
droite()
droite()
droite()
haut()
haut()
haut()
message = lire()
bas()
bas()
bas()
droite()
droite()
droite()
droite()
deverrouiller(message)
droite()
```

6. Un exemple d'instructions Python pour que BORDy sorte de la salle 5 :

```
droite()
droite()
droite()
bas()
bas()
bas()
bas()
message = lire()
haut()
haut()
haut()
haut()
droite()
droite()
deverrouiller(message)
droite()
droite()
droite()
haut()
haut()
message = lire()
bas()
bas()
droite()
droite()
deverrouiller(message)
droite()
```

7. En regardant les messages en bas à gauche, on s'aperçoit que la valeur stockée dans la variable `message` a changé. Les messages à lire étant aléatoires, la valeur stockée dans la variable `message` dépend de chaque élève. Le code de la première porte a été écrasé.

8. Le symbole de l'affectation est « = ». Comme on peut le voir dans les messages en bas à gauche, la variable `message` contient le premier code puis le deuxième, donc la variable `message` qui est de type chaîne de caractères ne peut contenir qu'une valeur à la fois.

9. L'ordre d'exécution des instructions Python est important, sinon BORDy est bloqué par les murs.

La correction dans l'éditeur BORDy est disponible à cette adresse : https://lienbordas.fr/740171_bordy1_BBQ.

Activité 2 p. 13 Comment structurer un programme à l'aide de fonctions ?

Capacité travaillée :

- Définitions et appels de fonctions.

Cette acticité présente la définition d'une fonction Python, ainsi que son appel avec et sans argument.

La question 3 demande à recopier la fonction `generer_code()` qui est définie dans le doc. C. Elle permet de voir un premier exemple de fonction avec un argument.

La question 4 demande de recopier une fonction sans argument afin de limiter le nombre de lignes du programme. La limitation du nombre de lignes oblige l'élève à utiliser la fonction du document B.

Les questions 5, 6 et 7 permettent de montrer l'intérêt de programmer en utilisant des fonctions.

Réponses aux questions :

2. Un exemple d'instructions Python pour que BORDy sorte de la salle 6 :

```
droite()
droite()
droite()
droite()
droite()
droite()
bas()
saisir_cable()
inverser_branchement()
certifier()
haut()
droite()
droite()
droite()
droite()
droite()
```

3. Un exemple d'instructions Python pour que BORDy sorte de la salle 7 :

```
def generer_code(login):
      code = login[::-1]
      return code
droite()
droite()
haut()
saisir_cable()
inverser_branchement()
certifier()
droite()
droite()
bas()
bas()
saisir_cable()
inverser_branchement()
certifier()
bas()
bas()
bas()
droite()
saisir_cable()
inverser_branchement()
certifier()
droite()
droite()
saisir_cable()
inverser_branchement()
certifier()
droite()
droite()
message = generer_code('bordy')
droite()
haut()
haut()
haut()
haut()
deverrouiller(message)
droite()
```

4. La limitation de la mémoire de BORDy (et donc du nombre de lignes du programme) empêche BORDy de réaliser sa mission.

5. En utilisant la fonction `reparer()` plutôt qu'en écrivant la séquence des trois instructions du doc. A, on réduit le nombre de lignes du programme, ce qui permet de le stocker dans la mémoire de BORDy.

6. Un exemple d'instructions Python pour que BORDy sorte de la salle 8 :

```
def generer_code(login):
      code = login[::-1]
      return code
def reparer():
      saisir_cable()
      inverser_branchement()
      certifier()
droite()
droite()
haut()
reparer()
droite()
droite()
bas()
bas()
reparer()
bas()
bas()
bas()
droite()
reparer()
droite()
droite()
reparer()
droite()
droite()
message = generer_code('bordy')
droite()
haut()
haut()
haut()
haut()
deverrouiller(message)
droite()
```

7. L'utilisation de fonctions permet de réduire le nombre de lignes dans un programme et d'éviter les erreurs liées à la réécriture d'instructions. Par exemple, si l'on veut modifier la séquence de trois instructions dans le programme précédent, il suffit de changer la définition de la fonction `reparer()`.

La correction dans l'éditeur BORDy est disponible à cette adresse : https://lienbordas.fr/740171_bordy2_RFC.

Activité 3 p. 14 Comment adapter un programme aux situations rencontrées ?

Capacité travaillée :

- Instructions conditionnelles.

Cette acticité permet de faire découvrir aux élèves les instructions conditionnelles. Sans l'ajout de ces instructions, BORDy débranche les prises déjà connectées.

La distribution des prises débranchées est aléatoire, donc les élèves sont obligés d'utiliser des instructions conditionnelles.

Réponses aux questions :

1. On va à l'adresse indiquée.

2. Avec le programme déjà présent dans l'interface, BORDy inverse le ou les branchements, donc le terminal ne peut pas s'allumer et la porte reste fermée.

3. La fonction `interrompu()` renvoie `True` ou `False` en fonction de l'état du câble. Cette fonction va donc permettre de tester l'état du câble avant d'inverser le branchement.

4. a. Dans la fonction `reparer_si()`, la condition est `interrompu()`.

b. L'instruction qui joue le rôle de *`bloc_instructions_1`* est `reparer()`.

c. L'instruction qui joue le rôle de *`bloc_instructions_2`* est `certifier()`.

d. L'instruction qui est exécutée dans tous les cas est `certifier()` car elle est appelée dans la fonction `reparer()`.

5. Un exemple d'instructions Python pour que BORDy sorte de la salle 9 :

```
def reparer():
      saisir_cable()
      inverser_branchement()
      certifier()

def reparer_si():
      if interrompu():
            reparer()
      else:
            certifier()
droite()
droite()
reparer_si()
droite()
droite()
reparer_si()
droite()
droite()
reparer_si()
droite()
droite()
droite()
droite()
droite()
```

6. Un exemple d'instructions Python pour que BORDy sorte de la salle 10 :

```
def reparer():
      saisir_cable()
      inverser_branchement()
      certifier()

def reparer_si():
      if interrompu():
            reparer()
      else:
            certifier()
droite()
droite()
reparer_si()
droite()
droite()
reparer_si()
droite()
bas()
reparer_si()
haut()
droite()
droite()
droite()
droite()
droite()
droite()
```

7. Une instruction conditionnelle dans un programme permet d'exécuter un bloc d'instructions en fonction d'une condition, ce qui correspond donc à un choix.

La correction dans l'éditeur BORDy est disponible à cette adresse :
https://lienbordas.fr/740171_bordy3_ICQ.

Activité 4 p. 15 Comment répéter une tâche jusqu'au succès ?

Capacité travaillée :

- Boucles non bornées.

L'objectif de cette acticité est de présenter les boucles non bornées. L'utilisation de boucles non bornées permettra à BORDy de ne pas tester toutes les prises une fois que la porte s'ouvre.

Réponses aux questions :

1. On va à l'adresse indiquée.

2. En début de programme, la porte est fermée donc la variable `porte_ouverte` contient la valeur `False`.

3. Un exemple d'instructions Python pour que BORDy répare les connexions débranchées l'une après l'autre tant que la porte n'est pas ouverte :

```
def retablir_connexion():
    while porte_ouverte != True:
        reparer_si()
```

4. Après l'exécution du programme, tous les câbles sont réparés donc la porte s'ouvre.

5. Si la porte ne s'ouvre jamais, on risque d'avoir une boucle infinie car la boucle non bornée s'exécutera en boucle.

6. Dès que BORDy a réparé tout le câble, il s'arrête et ne va pas jusqu'à la sortie de la salle. Il doit alors se déplacer jusqu'à la porte.

7. Un exemple d'instructions Python pour que BORDy sorte de la salle 11 :

```
def retablir_connexion():
    while porte_ouverte != True:
        reparer_si()
    while position() != 'J6':
        droite()
```

8. Une variable booléenne ne peut prendre que deux valeurs, `True` et `False`.

9. Dans un programme, une boucle non bornée permet de répéter une séquence d'instructions tant qu'une condition est vraie.

La correction dans l'éditeur BORDy est disponible à cette adresse :
https://lienbordas.fr/740171_bordy4_EGO.

Activité 5 p. 16 Comment répéter une tâche un certain nombre de fois ?

Capacité travaillée :

- Boucles bornées.

Le but de cette acticité est de présenter les boucles bornées. L'utilisation de boucles bornées permettra à BORDy de répéter des séquences d'instructions puisque le nombre de lignes est limité.

Réponses aux questions :

1. On va à l'adresse indiquée.

2. a. Un exemple d'instructions Python pour que BORDy sorte de la salle 12 :

```
def traverser():
      for i in range(10):
            droite()
traverser()
deverrouiller('perl')
droite()
```

b. La boucle a effectué 10 tours.

3. Un exemple d'instructions Python pour que BORDy sorte de la salle 13 :

```
def compter():
      cpt = 0
      for i in range(4):
            if interrompu():
                  cpt = cpt + 1
            droite()
            droite()
      return cpt #pour variable dans tableau de variables
droite()
droite()
nombre = compter()
deverrouiller(nombre)
droite()
```

4. Un exemple d'instructions Python pour que BORDy sorte de la salle 14 :

```
def retablir_connexion_14():
      for i in range(4):
            reparer_si()
            droite()
            droite()

droite()
droite()
retablir_connexion_14()
droite()
```

5. Un exemple d'instructions Python pour que BORDy sorte de la salle 15 :

```
def reparer():
    saisir_cable()
    inverser_branchement()
    certifier()

def reparer_si():
    if interrompu():
        reparer()
    else:
        certifier()

def retablir_connexion_15():
    for i in range(9):
        droite()
        reparer_si()
        bas()

retablir_connexion_15()
for i in range(5):
      haut()
droite()
droite()
```

6. Une boucle bornée permet de répéter une séquence d'instructions un certain nombre de fois connu à l'avance. Elle raccourcit le programme et limite les erreurs dues au copier-coller.

7. Une boucle « pour » est utilisée lorsqu'on sait à l'avance combien de fois on va répéter une séquence d'instructions, alors qu'une boucle « tant que » s'exécute tant qu'une condition est vraie.

La correction dans l'éditeur BORDy est disponible à cette adresse : https://lienbordas.fr/740171_bordy5_MSN.

Activité 6 p. 17 Comment utiliser un éditeur et une console Python ?

Capacité travaillée :

- Définitions et appels de fonctions.

L'objectif de cette activité est de prendre en main un éditeur Python avec sa console. Les manipulations sont à réaliser dans l'interface WebPython en allant sur ce lien : https://lienbordas.fr/740171_webpython1.

La question 2 montre l'utilité des boutons de l'interface présentés dans le doc. A. Les questions 3, 4 et 5 permettent de se familiariser avec la console et montrent l'intérêt d'écrire une fonction que l'on appelle à trois reprises.

Réponses aux questions :

1. On va sur le lien proposé.

2. a. Si on reproduit l'erreur de Jade, l'éditeur Python affiche une erreur de syntaxe :

```
  File "<script>", line 1
    def calculer(octets)
                        ^
SyntaxError: expected ':'
```

b. Une fois le code corrigé et après avoir cliqué sur le bouton rouge, le message qui s'affiche dans la console est : `Bravo !`

3. Pour déterminer le nombre d'adresses IPv4 distinctes, on appelle la fonction `calculer()` avec pour argument 4 :

```
>>> calculer(4)
4294967296
```

4. Pour déterminer le nombre d'adresses IPv6 distinctes, on appelle la fonction `calculer()` avec pour argument 16 :

```
>>> calculer(16)
340282366920938463463374607431768211456
```

5. Pour déterminer le nombre d'adresses IPv6 distinctes en moyenne pour chaque individu :

```
>>> calculer(16)/8000000000
4.253529586511731e+28
```

6. La console Python permet de tester et de déboguer une fonction, mais aussi d'appeler facilement celles écrites dans l'éditeur, ce qui simplifie le développement et le rend plus interactif.

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython1_EZF.

C. Description des exercices

Exercice 1 p. 23 Utiliser une variable

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Affectations, variables, séquences.

Cet exercice très simple a pour objectif de clarifier les types de variables, surtout le type chaîne de caractères avec des questions sur les valeurs `'7'` ou `'True'`.

1. La valeur `2` est un entier donc son type est `int`.

La valeur `'BORDy'` est une chaîne de caractères donc son type est `str`.

La valeur `3.14` est un nombre à virgule flottante donc son type est `float`.

La valeur `False` est un booléen donc son type est `bool`.

La valeur `-0.5` est un nombre à virgule flottante donc son type est `float`.

La valeur `'b'` est une chaîne de caractères donc son type est `str`.

La valeur `'7'` est une chaîne de caractères donc son type est `str`.

La valeur `'True'` est une chaîne de caractères donc son type est `str`.

2. Le symbole qui permet d'affecter une valeur à une variable est « = ».

3. La séquence d'instructions est la suivante :

```
a = 2
b = -3.0
c = True
```

Exercice 2 p. 23 Utiliser un tableau de suivi des variables

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Affectations, variables, séquences.

Cet exercice relativement simple permet de comprendre comment évolue la valeur d'une variable, en utilisant un tableau de variables comme dans le cours p. 21. L'emploi d'un tableau de variables peut être utile pour déboguer ou tout simplement comprendre comment fonctionne un programme.

1. Les noms des trois variables utilisées dans le programme sont `a`, `b` et `c`.

Le programme comporte 5 instructions.

2. Voici le tableau complété :

```
01 a = 3
02 b = 7
03 c = a
04 a = b
05 b = c
```

a	b	c
3		
3	7	
3	7	3
7	7	3
7	3	3

3. En examinant le tableau, on s'aperçoit que les valeurs des variables `a` et `b` ont été échangées. Avant l'exécution de la ligne 3, la variable `a` contient la valeur `3` et la variable `b` contient la valeur `7`. À la fin de l'exécution de la ligne 5, la variable `a` contient la valeur `7` et la variable `b` contient la valeur `3`.

Exercice 3 p. 23 Concaténation de chaînes

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Affectations, variables, séquences.

Cet exercice relativement simple a pour objectif de manipuler des chaînes de caractères dans la console Python.

1. On va sur le lien proposé.

2. Initialisation des trois variables :

```
a = 'ma'
b = 'ro'
c = 'ti'
```

3. Dans la console, on obtient :

```
>>> a + b
'maro'
>>> b + a
'roma'
>>> a + a
'mama'
>>> 2 * a
'mama'
>>> b + a + c
'romati'
```

4. L'instruction qui permet d'écrire le mot `'roti'` est `b + c`, ce qui donne dans la console :

```
>>> b + c
'roti'
```

5. L'instruction `len(b + a)` renvoie la longueur de la chaîne de caractères `'roma'` donc 4, comme on peut le voir dans la console :

```
>>> len(b + a)
4
```

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython2_UQY.

Exercice 4 p. 23 Calculer la mensualité d'un prêt personnel

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Définitions et appels de fonctions.

Cet exercice illustre l'utilité des fonctions, qui peuvent être appelées un grand nombre de fois avec divers arguments pour déterminer une valeur précise.

1. On va sur le lien proposé.

2. La fonction `mensualite()` possède 3 arguments. Le premier argument est le montant du prêt, le deuxième est le taux d'intérêt annuel et le troisième est la durée du prêt.

3. La valeur que renvoie l'appel de la fonction `mensualite(10000, 3, 5)` est `179.69` :

```
>>> mensualite(10000, 3, 5)
179.69
```

Cette valeur correspond à la mensualité d'un prêt de 10 000 euros d'une durée de 5 ans avec un taux d'intérêt annuel de 3 %.

4. En procédant par tâtonnement ou dichotomie, on obtient :

```
>>> mensualite(12783, 4, 6)
199.99
>>> mensualite(12784, 4, 6)
200.01
```

La somme maximale que peut emprunter Léa est donc de 12 783 euros.

5. D'après la réponse précédente, Léa peut acheter la voiture.

L'instruction qui permet de calculer le montant des intérêts qu'elle paiera est la suivante :

```
>>> mensualite(12500, 4, 6) * 12 * 6 - 12500
1580.3200000000015
```

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython3_SGK.

Exercice 5 p. 23 Un brin d'ADN

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Définitions et appels de fonctions.

Cet exercice montre l'utilité de la programmation pour automatiser des recherches en biologie.

1. On va sur le lien proposé.

2. L'appel de fonction `synthetiser(10)` renvoie une chaîne de 10 caractères, par exemple : `'AACGTTTGCT'`.

3. Un nouvel appel de fonction `synthetiser(10)` renvoie généralement une autre chaîne de 10 caractères comme : `'CGAAGACCCG'`.

4. L'instruction qui permet de stocker un brin d'ADN de longueur 50 dans une variable nommée `brin` est : `brin = synthetiser(50)`.

5. Voici un exemple de valeur stockée dans la variable `brin` :

```
>>> brin = synthetiser(50)
>>> brin
'GCGTTCTTGACGATAATCGTGAGTGTCCCCTACACTACTTAAACCGTCCC'
```

Avec cette chaîne de caractères, l'instruction `'ATTC' in brin` renvoie `False`.

```
>>> 'ATTC' in brin
False
```

Avec cette chaîne de caractères, l'instruction `'TT' in brin` renvoie `True`.

```
>>> 'TT' in brin
True
```

Cette instruction renvoie `True` si la chaîne de caractères recherchée est présente dans la chaîne de caractères stockée dans la variable `brin`. Un généticien pourrait donc l'exploiter pour rechercher automatiquement certaines séquences de nucléotides dans des brins d'ADN.

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython4_YYL.

Exercice 6 p. 24 Quelques fonctions bien utiles

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Instructions conditionnelles

Cet exercice montre l'intérêt de décomposer un problème en petites fonctions, ce qui rend le code réutilisable, plus clair et plus facile à maintenir. La question 4 peut être ainsi écrite à l'aide des deux fonctions précédentes qui ont été déjà testées avec le bouton ✅ de l'éditeur WebPython. C'est l'occasion d'expliquer aux élèves que décomposer un problème complexe en plusieurs petites fonctions réutilisables dans un autre programme est une bonne pratique de programmation.

1. On va sur le lien proposé.

2. Exemples de correction :

```
# correction de la question 1
def est_positif(n):
    return n >= 0


# autre correction de la question 1
def est_positif_bis(n):
    if n >= 0:
        return True
    else:
        return False
```

3. Exemples de correction :

```
# correction de la question 2
def est_pair(n):
    return n % 2 == 0


# autre correction de la question 2
def est_pair_bis(n):
    if n % 2 == 0:
        return True
    else:
        return False
```

4. Exemples de correction :

```
# correction de la question 3
def est_pair_positif(n):
    return est_positif(n) and est_pair(n)


# autre correction de la question 3
def est_pair_positif_bis(n):
    if n >= 0 and n % 2 == 0:
        return True
    else:
        return False
```

La correction dans l'éditeur WebPython est disponible à cette adresse :
https://lienbordas.fr/740171_webpython5_JDM.

Exercice 7 p. 24 Encore une mission pour BORDy !

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Instructions conditionnelles

Cet exercice permet de manipuler les instructions conditionnelles avec BORDy. Il est conseillé d'avoir traité les activités 1, 2 et 3 p. 12 à 14 au préalable.

La correction dans l'éditeur BORDy est disponible à cette adresse :
https://lienbordas.fr/740171_bordy7_SAV.

Exercice 8 p. 24 Vive le camping !

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Instructions conditionnelles

Cet exercice permet de manipuler les instructions conditionnelles en demandant aux élèves d'écrire une fonction Python.

1. On va sur le lien proposé.

2. Exemples de correction :

```
# correction de la question 2
def cout(nombre_pers):
    forfait = 100
    if nombre_pers < 5:
        return forfait + nombre_pers * 10
    else:
        return forfait + nombre_pers * 7


# autre correction de la question 2
def cout_bis(nombre_pers):
    if nombre_pers >= 5:
        prix = 7 * nombre_pers
    else:
        prix = 10 * nombre_pers
    return 100 + prix
```

3. Voici les appels dans la console WebPython :

```
>>> cout(4)
140
>>> cout(5)
135
>>> cout(8)
156
```

Donc pour 4 personnes, le coût est de 140 euros ; pour 5 personnes, il est de 135 euros ; et pour 8 personnes, il est de 156 euros.

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython6_ESV.

Exercice 9 p. 24 Cybersécurité

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Boucles bornées

1. On va sur le lien proposé.

L'appel de la fonction nécessaire pour générer un code pin aléatoire est `genere()`.

Exemple d'appel :

```
>>> genere()
'3088'
```

2. Avant le premier tour de la boucle for, la variable `pin` contient une chaîne de caractères vide :

`pin = ''`.

À la fin du premier tour de boucle, la variable `pin` contient une chaîne de caractères avec un seul caractère. Exemple : `'7'`.

3. À chaque appel de la fonction, ce programme effectue 4 tours de boucle puisque la boucle est une boucle bornée : `for i in range(4)`.

4. Exemple de tableau complété :

i	pin
Non initialisée	''
0	'6'
1	'68'
2	'688'
3	'6884'

5. Exemple de code généré par ce programme : `'6884'`.

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython7_ODP.

Exercice 10 p. 24 Réparations réfléchies pour BORDy !

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Boucles bornées

Cet exercice permet de manipuler les boucles avec BORDy. Il est conseillé d'avoir traité les activités 1 à 4 p. 12 à 15 au préalable.

La correction dans l'éditeur BORDy est disponible à cette adresse : https://lienbordas.fr/740171_bordy8_LEG.

Exercice 11 p. 25 Écologie

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Boucles bornées et non bornées

1. On va sur le lien proposé.

2. Exemple de correction :

```
# correction de la question 2
def population(actuel, n):
    for i in range(n):
        actuel = actuel * 0.8 + 100
    return actuel
```

3. Appel de la fonction dans la console :

```
>>> population(1000, 5)
663.84
```

Cette valeur correspond à l'estimation de la population restante de félins au bout de 5 ans pour une population initiale de 1 000 tigres.

4. Exemple de correction :

```
# correction de la question 4
def survie(actuel, seuil):
    annees = 0
    while actuel >= seuil:
        actuel = actuel * 0.8 + 100
        annees = annees + 1
    return annees
```

5. Appel de la fonction dans la console :

```
>>> survie(1500, 800)
6
```

Pour une population initiale de 1 500 tigres et un seuil de 800, l'espèce sera menacée au bout de 6 décennies.

6.

	Valeur stockée dans la variable		La condition `actuel >= seuil` vaut …
	`actuel`	`annee`	
Avant la boucle	`1000`	`0`	`True`
Après le 1er tour	`900`	`1`	`True`
Après le 2e tour	`820`	`2`	`True`
Après le 3e tour	`756`	`3`	`False`

La correction dans l'éditeur WebPython est disponible à cette adresse : https://lienbordas.fr/740171_webpython8_LVP.

Exercice 12 p. 25 Objectifs de production

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Boucles bornées et non bornées

1. On va sur le lien proposé.

2. Exemple de correction :

```
# correction de la question 2
def production(initiale, n):
    for i in range(n):
        initiale *= 1.05
    return initiale
```

3. L'appel de la fonction dans la console :

```
>>> production(100000, 5)
127628.15625
```

4. Exemple de correction :

```
# correction de la question 4
def limite(initiale, objectif):
    for i in range(1000):
        if production(initiale, i) > objectif:
            return i
```

La correction dans l'éditeur WebPython est disponible à cette adresse :
https://lienbordas.fr/740171_webpython9_DHP.

Exercice 13 p. 25 BORDy part en mission !

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Boucles bornées et non bornées

Le lien proposé permet d'accéder à des défis à réaliser avec BORDy. Il est conseillé d'avoir traité les activités 1 à 5 p. 12 à 16 au préalable.

La correction dans l'éditeur BORDy est disponible à cette adresse :
https://lienbordas.fr/740171_defis_NIC.

Exercice 14 p. 25 Programmer un jeu vidéo

Capacité travaillée :

- Écrire et développer des programmes pour répondre à des problèmes et modéliser des phénomènes physiques, économiques et sociaux.

Contenus :

- Boucles bornées et non bornées

Cette interface permet d'approfondir ses connaissances en programmation Python. L'élève dispose de toutes les indications nécessaires pour programmer le jeu vidéo dans l'interface et peut ainsi travailler en autonomie. Le découpage en fonctions facilite également le travail en groupes, chaque groupe prenant en charge une partie du code, ce qui peut être intéressant dans le cadre d'un mini-projet. En général, le travail de l'élève est sauvegardé automatiquement dans son navigateur, mais il vaut mieux lui conseiller de télécharger son code s'il souhaite le reprendre ultérieurement.

La correction dans l'interface est disponible à cette adresse :
https://lienbordas.fr/740171_cassebriques_JVUMMD.