Nom :	Prénom :	Classe :

NSI 1re — Programmation défensive #1 : les préconditions

Objectifs :
- Savoir définir “paramètre”, “argument”, “assertion”, “programmation défensive”, “précondition”.
- Savoir écrire une précondition.

“Paramètre” vs “Argument”

- Un **paramètre** est le nom de la variable locale[1] utilisée dans la définition d'une fonction.
- Un **argument** est la valeur passée à l'appel de la fonction.

Illustrons cela par un exemple :

```
1  def f(h):
2      return h ** 2
3
4  a = f(10)
5  b = f(4)
```

Dans ce programme :

- `h` est le paramètre de la fonction `f`.
- `10` et `4` sont les arguments passés à l'appel de la fonction.

➤ Exercice 1 — Identifier

Dans les programmes ci-dessous, identifier les paramètres des arguments :

N°	Python	Paramètre(s)	Argument(s)

1	`def plus_un(a):`
	`  return a + 1`
	`i = plus_un(57)`
		a	57

2	`def st(a, b):`
	`  return a - b`
	`w = st(90, 100)`
	`z = st(14, 8)`
		a et b	90 et 100
			puis 14 et 8

3	`u = 180`
	`def a(b):`
	`  return b * 3.14 * 81`
	`c = a(u)`
		b	La valeur de “u”
			(qui est de “180”)

1 Voir le cours « Portée d'une variable : variable locale, variable globale ».

Programmation défensive

La **programmation défensive** consiste à écrire du code en anticipant les erreurs possibles, en particulier celles liées aux **arguments** d'une fonction, et à vérifier que ces arguments respectent bien certaines conditions (baptisées **préconditions**) avant de continuer.

En NSI, cela se traduit souvent par l'utilisation d'**assertions**.

Assertion

Une assertion est un test (une sorte de condition) que le programme vérifie pendant l'exécution.

- Si le test est vrai, rien ne se passe, le programme poursuit son exécution.
- Si le test est faux, Python déclenche une erreur ("AssertionError") et interrompt le programme.

En Python une assertion s'écrit en utilisant le mot-clé `assert` suivi d'une affirmation.
Et si l'affirmation est fausse, Python interrompt immédiatement le programme.

Voici un exemple d'assertion :

```
1  a = 100
2  assert a > 100
3  print('Fin du programme')
```

Dans le programme ci-dessus, le message "Fin du programme" ne s'affiche pas.

Pourquoi ?
Parce qu'il existe une assertion (ligne 2) qui est fausse. Donc le programme s'interrompt ligne 2.

➤ Exercice 2 — Vérifier

Vérifier si les programmes ci-dessous sont intégralement exécutés, ou si une assertion en interrompt l'exécution.

N°	Python	Intégralement exécuté (oui / non)	Justifier pourquoi

1	`a = 10`
	`a = a - 5`
	`assert a > 5`
	`c = a + 10`
		Non	L'assertion est fausse.
			
			Ligne 3 : "a" vaut "5", ne peut pas être > "5".

2	`a = 10`
	`assert a > 5`
	`for _ in range(3):`
	`  a += 1`
	`assert a > 10`
	`print(a)`
		Oui	Les deux assertions sont vraies.
			
			Respectivement :
			
			- Ligne 2 : "a" vaut "10" et est bien > "5".
			- Ligne 5 : "a" vaut "13" et est bien > "10".

Assertion au message personnalisé

Notons qu'il est également possible d'afficher un message personnalisé, si une assertion venait à être fausse.

Ce programme contient une assertion sans message personnalisé :

```
1  a = 100
2  assert a > 200
3  b = a ** 3
```

Et voici le même programme, utilisant cette fois une assertion au message personnalisé :

```
1  a = 100
2  assert a > 200, "a doit avoir une valeur strictement supérieure à 200"
3  b = a ** 3
```

Ici Python affichera l'erreur « AssertionError: a doit avoir une valeur strictement supérieure à 200 ». Et la ligne 3 ne sera pas exécutée.

Précondition

On appelle **précondition** l'assertion qui permet de vérifier la valeur d'un argument, dans une fonction.

Par exemple, la fonction suivante ne possède aucune précondition :

```
1  def f(t):
2      return t[0]
```

Tandis que celle-ci en possède une :

```
1  def g(t):
2      assert len(t) > 0, "Le tableau doit contenir au moins un élément"
3      return t[0]
```

La précondition écrite à la ligne 2 permet de vérifier que l'argument ne soit pas un tableau vide.

Amusons-nous maintenant à tester cette précondition, en exécutant la fonction `g` de deux manières différentes :

```
4  >>> g( [18, 4] )
5  # Tout va bien, pas d'erreur ...
6
7  # Autre test, avec un autre argument :
8  >>> g( [] )
9  AssertionError: Le tableau doit contenir au moins un élément
```

Super, notre assertion a bien été déclenchée !

➤ **Exercice 3** — Écrire une précondition dans une fonction documentée

Dans les programmes ci-dessous, lire la documentation de la fonction puis écrire la précondition correspondante.
La précondition du premier programme est donnée en guise d'exemple.

N°	Python	Précondition (à compléter)

0	`def f(a):`
	`  """Renvoie le carré d'un nombre`
	
	`  Param :`
	`  - a (int) Entier > 10`
	`  """`
	`  assert ……`
	`  c = a ** 2`
	`  return c`
		assert a > 10, "Argument doit être > 10"

1	`def f(a, b):`
	`  """Renvoie la somme de “a” et “b”`
	
	`  Param :`
	`  - a (int) Entier > 0`
	`  - b (int) Entier > 0`
	`  """`
	`  assert ……`
	`  assert ……`
	`  return a + b`
		assert a > 0, "La valeur de « a » doit être supérieure à 0"
		
		assert b > 0, "La valeur de « b » doit être supérieure à 0"

2	`def f(t):`
	`  """Renvoie les deux`
	`  derniers éléments de “t”`
	
	`  Param :`
	`  - t (list) Tableau contenant`
	`  au moins deux éléments.`
	`  """`
	`  assert ……`
	`  return (t[-1], t[-2])`
		assert len(t) >= 2, "Le tableau doit contenir au moins deux éléments"

3	`def f(a, b):`
	`  """Renvoie “b” - “a”.`
	
	`  Précondition : le résultat doit`
	`  être supérieur à 0.`
	`  """`
	`  assert ……`
	`  return b - a`
		assert b - a > 0, "Le résultat doit être supérieur à zéro"

➤ **Exercice 4** — À partir d'un cas d'usage, écrire une fonction utilisant des préconditions

L'entreprise *Brousse-Eau-Banque* a fait appel à vous pour créer une fonction `debit` qui prend deux paramètres `v` et `s` (correspondant respectivement à une valeur à débiter, et au solde d'un compte bancaire).

Cette fonction renvoie le résultat du calcul `s - v` uniquement si les préconditions suivantes sont respectées :

N°	Précondition	Message affiché
1	La valeur à débiter doit être strictement supérieure à 0.	Débit incorrect
2	La valeur à débiter ne doit pas aller au delà de 3000.	Débit maximum dépassé
3	La valeur à débiter ne doit pas être plus grande que le solde.	Solde insuffisant

À vous de jouer :

```
1  def debit(v, s):
2      assert v > 0, "Débit incorrect"
3      assert v <= 3000, "Débit maximum dépassé"
4      assert v <= s, "Solde insuffisant"
5      return s - v
```

Pour aller plus loin

Y-a-t-il des limites et des risques à l'utilisation de préconditions ?

Les préconditions ne remplacent pas tout. Elles ne disent pas comment la fonction calcule, seulement dans quels cas on a le droit de l’appeler. Elles peuvent aussi alourdir le code si elles sont trop nombreuses, ou devenir gênantes si on vérifie systématiquement des choses déjà garanties ailleurs.

Le principal risque est de mal définir la précondition : trop large, elle laisse passer des cas dangereux ; trop stricte, elle bloque des usages pourtant valides. Si les préconditions ne sont pas bien documentées, l'utilisatrice ou l’utilisateur de la fonction peut les ignorer et provoquer des erreurs difficiles à comprendre.

Un autre risque est de croire que l’assertion suffit à elle seule : elle aide à détecter un problème, mais elle ne remplace ni une bonne conception, ni des tests complets.