Déboguer un programme

La phase de débogage d'un programme, qui consiste à rechercher les erreurs de programmation, est très gourmande en temps. Ceci est particulièrement vrai avec Python dont le typage dynamique repousse la découverte des fautes au moment de l'exécution.

1. Le traceback de Python

Lorsque l'interpréteur Python rencontre un problème, une **exception** est levée. Si elle n'est pas capturée, cette exception provoque l'arrêt du programme et l'affichage d'un message appelé traceback. Ce message permet de connaître la nature et le contexte de l'incident.

```
>>> t = [1, 1, 2, 5, 14, 42, 1321]
>>> t[12]
Traceback (most recent call last):
  File "<stdins>", line 1, in <module>
IndexError: list index out of range
```

Le tableau suivant donne quelques exceptions courantes, observables en utilisant les structures de base de Python.

NameError	accès à une variable inexistante
IndexError	accès à un indice invalide d'un tableau
KeyError	accès à une clé inexistante d'un dictionnaire
ZeroDivisionError	division par zéro
TypeError	opération appliquée à des valeurs incompatibles
ValueError	Argument de valeur incompatible mais de bon type

2. Signaler un problème avec une exception

Il est possible de déclencher directement toutes ces exceptions (on dit « lever une exception ») avec l'opération **raise** de Python.

```
>>> raise IndexError('indice trop grand')
Taceback (most recent call last) :
  File "<stdins>", line 1, in <module>
IndexError : indice trop grand
```

Cette opération s'écrit en faisant suivre le mot-clé **raise** du nom de l'exception à lever, lui-même suivi entre parenthèses d'une chaîne de caractères donnant des informations sur l'erreur signalée.
Le message affiché lorsqu'une exception interrompt un programme et donne un aperçu de l'état de la pile d'appels au moment où l'exception à été levée.

Interfaces et exceptions

Les exceptions peuvent être utilisées en particulier dans les fonctions formant l'interface d'un module, pour signaler à un utilisateur du module toute utilisation incorrecte de ces fonctions. L'interface mentionnera dans ce cas quelles exceptions spécifiques sont levées et dans quelles conditions.

Rattraper une exception

Les exceptions levées par un programme peuvent avoir plusieurs causes. Certaines traduisent, des erreurs du programme : celles qui sont imprévues et leurs conséquences ne sont par définition pas maîtrisées. Dans ces conditions, interrompre l'exécution du programme est légitime. D'autres exceptions, en revanche, s'inscrivent dans le fonctionnement normal du programme : elles correspondent à des situations connues, exceptionnelles mais possibles.
Pour mettre en place ce comportement alternatif, il faut rattraper cette exception, c'est-à-dire l'intercepter avant que l'exécution du programme ne soit définitivement abandonnée, avec la construction suivante.

```
try :
    x = int (input ('Entrer un jour'))

except ValueError :
    print ('Entrer un entier valide')
```

Le mot-clé **try** suivi du symbole **:** (deux-points) introduit un premier bloc de code, puis le mot-clé **except** suivi du nom de l'exception et du symbole **:** précède un deuxième bloc de code. On qualifiera le premier bloc de normal et le second d'alternatif.

L'exécution d'une telle instruction procède comme suit. On exécute d'abord le bloc de code normal. Si l'exécution de ce bloc s'achève normalement, sans lever d'exception, alors le bloc alternatif est ignoré et on passe directement aux instructions suivantes. Si à l'inverse une exception est levée dans l'exécution du bloc normal, alors l'exécution de ce bloc est immédiatement interrompue et le nom de l'exception levée est comparé avec le nom précisé à la ligne **except**. Si les noms correspondent, l'exception est rattrapée et on exécute le bloc de code alternatif avant de passer à la suite. Sinon, l'exception est propagée et le programme s'interrompt.