Premiers pas avec PyCharm

Créer un projet

Nous supposons ici que PyCharm est déjà installé.

Lien vers la documentation exhaustive de PyCharm

Lorsque vous lancez PyCharm pour la première fois, une fenêtre apparaît:

- Cliquez sur "Create New Project"

Sur la fenêtre suivante:

- Avec le champ "Location", choisissez ensuite l'emplacement de votre dossier projet et son nom.
- Avec le champ "Interpreter" choisissez votre interpréteur python, par exemple : "Python 3.4" (si python3 n'est pas installé, regardez la section suivante)

Installer un interpréteur python3

Pour rajouter un nouvel interpréteur (parfois, Python3 n'est pas proposé), il faut cliquer sur le petit engrenage (à droite de la barre) & choisir "add local". Il faut attendre un peu le temps que pycharm scanne votre disque dur à la recherche des versions de python. Une fois ceci fait, vous pourrez facilement le fichier python3 (puis cliquer dessus).

Créer un fichier Python

Nous voilà arrivés à l'interface de l'IDE. Comme vous pouvez le constater il n'y a aucun fichier dans le projet.

Pour créer un nouveau fichier python:

- **Menu "File" » "New" (ou Alt + Inser") » "Python File**". Vous n'avez plus qu'a rentrer le nom.

Vous pouvez alors écrire

```
print("Hello World!")
```

dans ce fichier et l'exécuter à l'aide du menu "Run". Cela vous affiche

```
Hello World!
```

dans la fenêtre console située au bas de l'éditeur.

Les Menus

- **Navigate** permet de rechercher des fichers, classes etc
- **Refactor** permet de modifier des noms de modules dans tout un projet, des noms de variables dans une fonction ou un fichier, à l'aide d'une seule manipulation.
- **Run** permet d'executer un fichier ou tout le projet.

Le menu File

Le menu File permet bien évidemment d'ouvrir/fermer un fichier, un projet... Cependant c'est aussi là que se trouve les options de configuration de PyCharm dans le sous-menu "**Settings**" ou bien à l'aide de la combinaison "**Ctrl + Alt + S**".

Voici quelques manipulations qui nous simplifieront la vie plus tard:

1. Allez dans "Settings" » Editor » Code Style » File and Code Template » Python Script
2. Allez dans "Settings" » Editor » General » Appearance puis cochez "**Show line numbers**"

La première manipulation permet de dire à l'interpréteur Python qu'on utilise des caractères UTF-8. Sans cette précision, des erreurs apparaissent à l'exécution si vous utilisez des caractères tels que "é", "è", "à" par exemple.

Des raccourcis qui simplifient la vie

PyCharm est préconfiguré avec de nombreux raccourcis. Vous pouvez y accéder dans "Settings" » Appearance and Behavior » Keymap. Vous pouvez alors modifier les combinaisons ou bien consulter celles qui existent.

Quelques raccourcis par défaut utiles:

- Commenter une ligne ou un bloc: "Ctrl + /" (Utiliser le "/" du pavé numérique)
- Rechercher "Ctrl + F"
- Exécuter le fichier actuel: "Maj + F10"
- Exécuter tout le projet: "Alt + Maj + F10"

Ajouter un environnement d'exécution

Les environnement d'exécutions permettent d'exécuter des programmes ou des tests python. On les trouve soit grace à la barre de menu, item **Run**, ou en haut à droite de la fenêtre (son nom puis un petit triangle vert pour exécuter l'environnement).

Pour créer un nouvel environnement d'exécution:

1. créez un nouveau contexte d'exécution dans le menu run > edit configuration...

2. la fenêtre qui s'est ouverte contient tous les contextes d'exécution de votre projet. Cliquez sur le + en haut à gauche de la fenêtre pour en créer un nouveau.
3. choisissez **python** ou **python tests**, paramètrez votre environnement en choisissant le script à exécuter (on pourra également nommer son environnement).
4. cliquer sur **OK** pour créer l'environnement.

Dans la fenêtre de gestion des contextes, ne modifiez pas les contextes par défaut. Ce sont des templates.

Pour plus d'information, reportez vous à la documentation de pycharm.

Les tests

Nous devons être certains que toutes les méthodes, fonctions ou modules que nous créons soient corrects. On écrira donc des tests pour être moralement sûrs que nos programmes fonctionnent (la plupart du temps une preuve de code est illusoire).

Pour éviter de retaper tous ces tests à chaque modification du code (ce qui arrive souvent lorsque un algorithme ou une application est utilisée longtemps) ou à chaque découverte de bug, ils sont conservés dans un fichier à part. Ceci nous permettra d'exécuter ces tests à loisir (c'est à dire très souvent) et d'être sûrs que **tous** les tests seront exécutés. Ces tests sont dit unitaires et sont essentiels dans toutes les pratiques courantes de code.

Environnement de tests avec pyCharm

De nombreux environnements de tests existent pour pycharm, nous allons utiliser py.test.

Premier exemple

Créez un nouveau projet avec pycharm que l'on pourra appeler `essai_tests`, puis ajoutez-y un fichier que vous nommerez `aide_mathematiques.py`. Ce fichier contiendra le code suivant :

```
def double(entier):
  return 2 * entier
```

Pour tester ce code, j'imagine que si les deux conditions suivantes sont remplies :

- double(0) vaut 0,
- double (21) vaut 42

Ma méthode sera exacte.

On utilise le mot clé assert pour créer notre fonction de test.

Les fonctions de tests doivent toutes commencer par `test_`

Ajouter la méthode ci-après à votre fichier :

```
def test_double():
  assert double(0) == 0
  assert double(21) == 42
```

et executez là :

```
test_double()
```

Si tout s'est passé comme prévu, il ne s'est rien passé. Normal, l'`assert` était vérifié. Changez un des `assert` de la fonction `test_double` pour que le résultat soit faux (par exemple `assert double(0) == 7`). Le programme doit maintenant s'arrêter sur une exception. Chez moi, j'obtiens ça :

```
Traceback (most recent call last):
  File
"/Users/francois/Documents/pycharm/essai_tests/aide_mathematiques.py", line
10, in <module>
    test_double()
  File
"/Users/francois/Documents/pycharm/essai_tests/aide_mathematiques.py", line
6, in test_double
    assert double(0) == 7
AssertionError
```

Ainsi, si tout se passe bien, nos tests sont passés, si le programme s'arrête sur une exception de type `AssertionError`, nos tests ne correspondent pas à la réalité. Nous sommes en face d'un bug (qu'il faut corriger).

Séparer code et tests

Placez la fonction de test (et son exécution) dans un fichier que vous nommerez `test_aide_mathematiques.py`.

Faites en sorte qu'il s'exécute sans problème (attention aux `import`).

On séparera toujours les tests du code. Tout fichier de test commence par `test_`.

Utilisation de l'environnement de test

Nous allons demander à l'environnement py.test d'exécuter nos tests. Il nous donnera plus d'informations sur les tests réussis ou échoués (une application normale contient des centaines de

tests).

Commencez par supprimer l'exécution de `test_double` dans le fichier `test_aide_mathematiques.py`.


Un fichier de test ne doit contenir que des fonctions.

Puis nous allons demander à pycharm d'exécuter `test_aide_mathematiques.py` à l'aide de notre environnement de test. Pour cela, suivez les instructions de la partie Ajouter un environnement d'exécution et créez une configuration *pyhton test* > *py.test*. Ici, les paramètres dont nous aurons besoin sont :

- le champ `name`, qui donne un nom à notre contexte. Par exemple `mes tests`
- le champ `target`, qui spécifie quel script utiliser. Cliquez tout à droite de ce champ sur un petit bouton avec **…** puis choisissez le fichier `test_aide_mathematiques.py`

Une fois ceci configuré, cliquez sur le bouton OK.

Un nouvel environnement de test est créé dans l'onglet `run`. Exécutez le. Vous devriez voir une nouvelle fenêtre en bas de l'écran pycharm apparaître et vos tests s'exécuter. Si tout s'est bien passé, une barre verte doit apparaître.

Pour finir cette partie :

- séparez votre fonction de tests en 2 fonctions (chaque fonction de test ne doit contenir qu'une chose à tester, donc a priori qu'un seul `assert`,
- exécutez votre nouvel environnement
- ajoutez une fonction de test qui plante. Exécutez votre environnement de test. Voyez la barre rouge. Supprimez ce test non valide.

Les tests en ligne de commande

La bibliothèque py.test peut directement s'exécuter depuis le terminal. En supposant que votre fichier de test s'appelle `test_aide_mathematiques.py` et que vous vous trouviez dans le bon répertoire, la commande : `python3 -m pytest test_aide_mathematiques.py` va exécuter vos tests, comme vous le feriez depuis yCharm.

Rédacteurs

- Augustin Agbo-Kpati
- François Brucker