



**POLYTECHNIQUE  
MONTRÉAL**

# LOG3430 - Méthodes de test et de validation du logiciel

## **Lab 1 : Couverture de test**

Soumis par :

Coéquipier 1 : Harry LAW-YEN - 2064104

Coéquipier 2 : Gabriel URIZA - 2195379

Équipe 69

Groupe 01

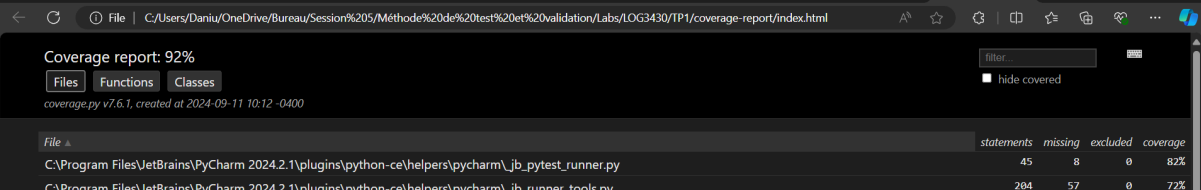
Dimanche 22 septembre 2024

Département de Génie Informatique et Génie Logiciel  
Polytechnique Montréal

**Important !** : Il est nécessaire d'inclure dans le rapport les captures d'écran illustrant les résultats des rapports de couverture pour chaque question. Pour la question 7, veuillez également intégrer des captures d'écran du code que vous avez rédigé.

**(a) Question 1** : Utilisez l'IDE PyCharm pour générer un rapport de couverture de code (Run 'pytest in tests' with Coverage, puis Generate Coverage Report). Quelle est la couverture totale du code pour Flask ? Vous devez aussi soumettre le dossier complet du rapport de couverture sous la forme de fichier zip nommé Q1.zip.

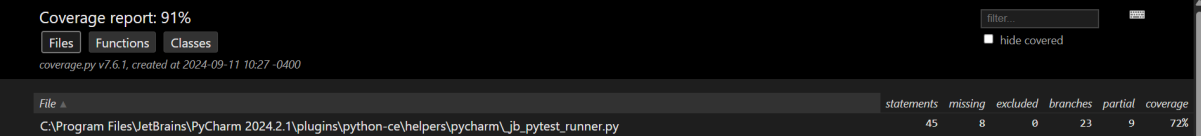
Rep :



The screenshot shows a web browser window displaying a Coverage report. The title is "Coverage report: 92%". Below the title, there are tabs for "Files", "Functions", and "Classes". The "Files" tab is selected. The report shows two files with their respective coverage statistics:

File	statements	missing	excluded	coverage
C:\Program Files\JetBrains\PyCharm 2024.2.1\plugins\python-ce\helpers\pycharm\jb_pytest_runner.py	45	8	0	82%
C:\Program Files\JetBrains\PyCharm 2024.2.1\plugins\python-ce\helpers\pycharm\jb_runner_tools.py	204	57	0	72%

**(b) Question 2 :** Utilisez l'IDE PyCharm pour activer la couverture des branches (Settings - Build, Execution, Deployment - Coverage) et générer un nouveau rapport de couverture de code. Quelle est la couverture totale du code pour Flask ? Y a-t-il une différence avec le rapport précédent ? Expliquez pourquoi il y a/n'y a pas de différence. Vous devez aussi soumettre le dossier complet du rapport de couverture sous la forme de fichier zip nommé Q2.zip.



Coverage report: 91%

Files Functions Classes

coverage.py v7.6.1, created at 2024-09-11 10:27 -0400

File	statements	missing	excluded	branches	partial	coverage
C:\Program Files\JetBrains\PyCharm 2024.2.1\plugins\python-ce\helpers\pycharm\jb_pytest_runner.py	45	8	0	23	9	72%

On remarque une différence de 1%, cette différence semble être dû au fait que le premier rapport traite la couverture de ligne (les noeuds dans le graph de flux de contrôle) et que le deuxième traite la couverture de branche. Il est donc fort possible que certaines opérations ternaires exécutées sur une seule ligne ne sont pas exécutées avec toutes leurs possibilités (cad que la première couverture ne couvre pas nécessairement toutes les branches).

**(c) Question 3 :** En utilisant la méthode de votre choix, excluez un fichier du processus de génération du rapport de test (sans le supprimer). Quel fichier avez-vous exclu (nom et répertoire du fichier) et comment l'avez-vous exclu du processus ? Comment les résultats changent-ils par rapport au dernier rapport (conservez l'option 2 pour avoir la couverture des branches) ? Vous devez aussi soumettre le dossier complet du rapport de couverture sous la forme de fichier zip nommé Q3.zip.

On a décidé d'exclure le fichier `blueprints.py` (chemin : `src/flask/blueprints.py`) dans cette question.

```
// Commande principale pour cette question qui utilise coverage.py
coverage run --branch --source=src/flask --omit=src/flask/blueprints.py -m
pytest

// Commande supplémentaire pour créer un rapport html
coverage html -d htmlReport3
```

Voici les changements avant (gauche) / après (après) :

Coverage report: 91%						
Files Functions Classes						
coverage.py v7.6.1, created at 2024-09-19 17:38 -0400						
File	statements	missing	excluded	branches	partial	coverage
src\flask\__init__.py	48	4	0	2	1	90%
src\flask\__main__.py	2	2	0	0	0	0%
src\flask\app.py	396	26	5	180	17	91%
src\flask\blueprints.py	34	8	2	10	2	68%
src\flask\cli.py	440	64	0	216	23	85%
src\flask\config.py	116	4	0	54	4	95%
src\flask\ctx.py	152	20	5	38	7	84%
src\flask\debughelpers.py	90	12	0	41	12	80%
src\flask\globals.py	14	0	7	0	0	100%
src\flask\helpers.py	121	6	2	48	5	92%
src\flask\json\_init_.py	25	3	2	8	2	85%
src\flask\json\provider.py	66	6	3	16	2	90%
src\flask\json>tag.py	135	0	0	30	0	100%
src\flask\logging.py	30	0	2	15	0	100%
src\flask\sansio\app.py	236	8	5	98	4	96%
src\flask\sansio\blueprints.py	226	4	2	118	8	97%
src\flask\sansio\scaffold.py	213	12	2	90	5	93%
src\flask\sessions.py	118	3	5	20	0	98%

Coverage report: 92%						
Files Functions Classes						
coverage.py v7.6.1, created at 2024-09-19 17:36 -0400						
File	statements	missing	excluded	branches	partial	coverage
src\flask\__init__.py	48	4	0	2	1	90%
src\flask\__main__.py	2	2	0	0	0	0%
src\flask\app.py	396	26	5	180	17	91%
src\flask\cli.py	440	64	0	216	23	85%
src\flask\config.py	116	4	0	54	4	95%
src\flask\ctx.py	152	20	5	38	7	84%
src\flask\debughelpers.py	90	12	0	41	12	80%
src\flask\globals.py	14	0	7	0	0	100%
src\flask\helpers.py	121	6	2	48	5	92%
src\flask\json\_init_.py	25	3	2	8	2	85%
src\flask\json\provider.py	66	6	3	16	2	90%
src\flask\json>tag.py	135	0	0	30	0	100%
src\flask\logging.py	30	0	2	15	0	100%
src\flask\sansio\app.py	236	8	5	98	4	96%
src\flask\sansio\blueprints.py	226	4	2	118	8	97%
src\flask\sansio\scaffold.py	213	12	2	90	5	93%
src\flask\sessions.py	118	3	5	20	0	98%
src\flask\signals.py	13	0	0	0	0	100%

**(d) Question 4 :** Veuillez expliquer, étape par étape, comment utiliser coverage.py pour calculer la couverture de test de Flask sans IDE (utilisez l'option pour branch coverage). Justifiez chaque terme de la commande que vous avez utilisée. Vous devez aussi soumettre le dossier complet du rapport de couverture sous la forme de fichier zip nommé Q4.zip.

```
// Commande principale pour cette question qui utilise coverage.py
coverage run --branch --source=src/flask -m pytest

// Commande supplémentaire pour créer un rapport html
coverage html -d htmlReport4
```

**coverage :** C'est l'outil de couverture de code que nous voulons utiliser.

**run :** Ça indique que nous voulons exécuter un script Python sous le contrôle de coverage.py.

**--branch :** Permet d'activer la couverture des branches.

**--source=src flask :** Indique le répertoire à couvrir. Cela exclut automatiquement tout ce qui n'est pas dans le chemin spécifié, y compris les fichiers de test dans d'autres répertoires.

**-m pytest :** Exécute pytest pour lancer les tests.

**(e) Question 5 :** Veuillez expliquer, étape par étape, comment utiliser pytest-cov pour calculer la couverture de test de Flask sans IDE (utilisez l'option pour branch coverage). Justifiez chaque terme de la commande que vous avez utilisée. Vous devez aussi soumettre le dossier complet du rapport de couverture sous la forme de fichier zip nommé Q5.zip.

```
pytest --cov=src --cov-branch tests/ --cov-report=html:htmlReport5
```

**pytest** : Lance les tests avec le framework pytest.

**--cov=src** : Cible le répertoire src pour mesurer la couverture de code. Cela signifie que pytest-cov mesurera uniquement la couverture du code source dans ce répertoire. pytest-cov en a également besoin pour générer un rapport html.

**--cov-branch** : Active la couverture des branches pour vérifier que toutes les branches conditionnelles sont couvertes au moins une fois (ne check pas les conditions élémentaires dans les instructions).

**tests/** : Spécifie le dossier où pytest doit rechercher les tests à exécuter. En utilisant tests/, pytest exécutera tous les tests dans ce dossier.

**--cov-report=html:htmlReport5** : Génère un rapport HTML de la couverture et le stocke dans le dossier htmlReport5.

**(f) Question 6 :** Veuillez analyser et discuter les éventuelles divergences en termes de couverture de lignes entre les deux méthodes récemment abordées. Quels sont les types de lignes couvertes par l'un mais pas par l'autre ? (Nommez-en 3). Expliquez pourquoi ces lignes sont couvertes/ignorées par ces outils ?

**Indice :** Comparer deux rapports de couverture que vous avez générés dans les questions précédentes pour un même fichier. (Veuillez soumettre des captures d'écran appuyant vos réponses).

Coverage avec coverage.py

Coverage avec pytest-cov

Coverage report: 91%

FilesFunctionsClasses

coverage.py v7.6.1, created at 2024-09-18 10:27 -0400

File	statements	missing	excluded	branches	partial	coverage
src/flask\__init__.py	48	4	0	2	1	90%
src/flask\__main__.py	2	2	0	0	0	0%
src/flask\app.py	396	26	5	180	17	91%
src/flask\blueprints.py	34	8	2	10	2	68%
src/flask\cli.py	440	64	0	216	23	85%
src/flask\config.py	116	4	0	54	4	95%
src/flask\ctx.py	152	20	5	38	7	84%
src/flask\debughelpers.py	90	12	0	41	12	80%
src/flask\globals.py	14	0	7	0	0	100%
src/flask\helpers.py	121	6	2	48	5	92%
src/flask\json\__init__.py	25	3	2	8	2	85%
src/flask\json\provider.py	66	6	3	16	2	90%
src/flask\json\tag.py	135	0	0	30	0	100%
src/flask\logging.py	30	0	2	15	0	100%
src/flask\sansio\app.py	236	8	5	98	4	96%
src/flask\sansio\blueprints.py	226	4	2	118	8	97%
src/flask\sansio\scaffold.py	213	12	2	90	5	93%
src/flask\sessions.py	118	3	5	20	0	98%
src/flask\signals.py	13	0	0	0	0	100%
src/flask\templating.py	110	6	4	32	8	89%
src/flask\testing.py	112	4	4	40	3	95%
src/flask\typing.py	21	0	4	0	0	100%
src/flask\views.py	54	1	0	22	1	97%
src/flask\wrappers.py	57	1	2	24	1	98%
Total	2829	194	56	1102	105	91%

Coverage report: 91%

FilesFunctionsClasses

coverage.py v7.6.1, created at 2024-09-12 10:38 -0400

File	statements	missing	excluded	branches	partial	coverage
src/flask\__init__.py	48	4	0	2	1	90%
src/flask\__main__.py	2	2	0	0	0	0%
src/flask\app.py	396	26	5	180	17	91%
src/flask\blueprints.py	34	8	2	10	2	68%
src/flask\cli.py	440	64	0	216	23	85%
src/flask\config.py	116	4	0	54	4	95%
src/flask\ctx.py	152	20	5	38	7	84%
src/flask\debughelpers.py	90	12	0	41	12	80%
src/flask\globals.py	14	0	7	0	0	100%
src/flask\helpers.py	121	6	2	48	5	92%
src/flask\json\__init__.py	25	3	2	8	2	85%
src/flask\json\provider.py	66	6	3	16	2	90%
src/flask\json\tag.py	135	0	0	30	0	100%
src/flask\logging.py	30	0	2	15	0	100%
src/flask\sansio\app.py	236	8	5	98	4	96%
src/flask\sansio\blueprints.py	226	4	2	118	8	97%
src/flask\sansio\scaffold.py	213	12	2	90	5	93%
src/flask\sessions.py	118	3	5	20	0	98%
src/flask\signals.py	13	0	0	0	0	100%
src/flask\templating.py	110	6	4	32	8	89%
src/flask\testing.py	112	4	4	40	3	95%
src/flask\typing.py	21	0	4	0	0	100%
src/flask\views.py	54	1	0	22	1	97%
src/flask\wrappers.py	57	1	2	24	1	98%
Total	2829	194	56	1102	105	91%

Malheureusement, nous ne remarquons pas de différences dans les screenshots... On ne peut rien dire juste en se basant sur nos rapports de couverture.

Nous allons donc comparer théoriquement les différences qu'on aurait dû constater avec la documentation :

1) Couverture des branches (lignes exécutées vs branches prises)

**coverage.py avec --branch** : Cet outil prend en compte les branches conditionnelles dans le code, comme les alternatives dans les instructions if et les

boucles while. Il marque une ligne comme partiellement couverte si toutes les branches possibles n'ont pas été exécutées.

**pytest-cov** : Utilisant coverage.py, il supporte également la couverture des branches. Cependant, grâce à son intégration avec pytest, il gère mieux les tests distribués et les sous-processus. Il n'offre pas de traitement supplémentaire spécifique des branches par rapport à coverage.py.

## 2) Fichiers Non Exécutés Directement

**Couvert par pytest-cov** : Grâce à ses options de configuration, pytest-cov peut forcer la couverture de fichiers qui ne sont pas exécutés directement lors des tests (par exemple, ceux qui sont chargés via des sous-processus ou des scripts externes).

**Ignoré par coverage.py** : Ce dernier se concentre sur les fichiers explicitement exécutés pendant le test et peut ignorer les fichiers utilisés indirectement.

## 3) Tests Parallélisés ou Paramétrés

**Couvert par pytest-cov** : En utilisant pytest-xdist pour exécuter des tests en parallèle ou paramétrés, pytest-cov combine automatiquement les résultats de couverture des processus parallèles.

**Ignoré ou partiellement couvert par coverage.py** : Sans configuration spécifique, coverage.py peut avoir du mal à combiner les résultats des processus parallèles ou des sous-processus, entraînant des omissions dans la couverture.



**(g) Question 7 :** Vous devez rédiger deux nouveaux tests afin d'améliorer la couverture de test de la bibliothèque Flask. Veuillez détailler les tests que vous avez créés. Utilisez la méthode de votre choix, parmi celles mentionnées dans les questions précédentes relatives à la couverture de test, pour montrer quel(s) critère(s) de couverture de test vos nouveaux tests ont amélioré(s). Vous devez aussi soumettre le dossier complet du rapport de couverture ainsi que le repo Flask complet sous la forme de fichier zip nommé Q7.zip.

The image contains two screenshots of a code editor. The top screenshot shows the `TestSendFile` class in `test_helpers.py` with two test methods: `test_static_file_with_time_delta` and `test_send_from_directory`. The bottom screenshot shows the `Blueprint` class in `blueprints.py` and the `test_blueprints.py` file with tests for `test_blueprint_specific_user_error_handling`, `test_errors_blueprints`, and `test_blueprint_app_error_handling`.

```

class TestSendFile:
    def test_static_file_with_time_delta(self, app, req_ctx):
        flask.current_app.config["SEND_FILE_MAX_AGE_DEFAULT"] = timedelta(0)
        rv = flask.send_file("static/index.html")
        assert rv.status_code == 200
        rv.close()

    def test_send_from_directory(self, app, req_ctx):
        app.root_path = os.path.join(

class Flask(App):
    def get_send_file_max_age(self, filename: str | None) ->
        if isinstance(value, timedelta):
            return int(value.total_seconds())
        return value # type: ignore[no-any-return]

    def send_static_file(self, filename: str) -> Response:
        """The view function used to serve files from
        :attr: 'static_folder'. A route is automatically registered

```

```

class Blueprint(SansioBlueprint):
    def send_static_file(self, filename: str) -> Response:
        # send_file only knows to call get_send_file_max_age on the app,
        # call it here so it works for blueprints too.
        max_age = self.get_send_file_max_age(filename)
        return send_from_directory(
            t.cast(str, self.static_folder), filename, max_age=max_age
        )

    def open_resource(
        self, resource: str, mode: str = "rb", encoding: str | None = "utf-8"
    ) -> t.IO[t.AnyStr]:
        """Open a resource file relative to :attr: 'root_path' for reading. The
        blueprint-relative equivalent of the app's :meth: '~Flask.open_resource'
        method.

        :param resource: Path to the resource relative to :attr: 'root_path'.
        :param mode: Open the file in this mode. Only reading is supported,
            valid values are "r" (or "rt") and "rb".
        :param encoding: Open the file with this encoding when opening in text
            mode. This is ignored when opening in binary mode.

        .. versionchanged:: 3.1
            Added the "encoding" parameter.

        """
        if mode not in {"r", "rt", "rb"}:
            raise ValueError("Resources can only be opened for reading.")

def test_blueprint_specific_user_error_handling(app, client):
    return "bam"

blue.register_error_handler(MyFunctionException, my_function_exception_handler)

@blue.route("/decorator")
def blue_deco_test():
    raise MyDecoratorException()

@blue.route("/function")
def blue_func_test():
    raise MyFunctionException()

app.register_blueprint(blue)

assert client.get("/decorator").data == b"bam"
assert client.get("/function").data == b"bam"

def test_errors_blueprints(app, client):
    blueprint = flask.Blueprint("error_opening_resources", __name__)
    try:
        blueprint.open_resource("static/index.html", "not a mod")
    except Exception as e:
        assert e.args[0] == "Resources can only be opened for reading."

def test_blueprint_app_error_handling(app, client):

```

Ces tests améliorent la couverture de branches et d'instruction, ainsi que la couverture de condition.

Le premier test vise à entrer à l'intérieur du nœud gardé par la condition `isInstance(value, timedelta)` qui retourne la valeur `int` si l'objet est de type. Pour y parvenir, il faut faire varier la condition booléenne de façon à ce qu'elle soit fausse. En effectuant cette modification à l'intérieur du nouveau test, nous augmentons donc la couverture de condition en même temps que la couverture nœuds et donc de branches.

C'est le même cas pour le deuxième test. La différence est que la condition est différente (`mode not in {"r", "rt", "rb"}`).