



Flask avec Visual Studio

I. CREER UN PROJET FLASK SOUS VISUAL STUDIO	3
1. CONFIGURATION DE L'ENVIRONNEMENT :	3
2. PREMIER PROGRAMME	4
a. Lancement.....	5
b. Dans un navigateur : http://127.0.0.1:5000/	5
3. UN AUTRE CODE PLUS ABOUTI	5
a. Code flask.....	6
b. Navigateur :	7
4. AVEC UN FICHIER DE CONFIGURATION C'EST MIEUX.....	7
a. Création du fichier de lancement	7
5. DEBUGGER	9
a. Point d'arrêt.....	9
b. Erreur	10
II. PREMIERS ESSAIS :	11
1. PROGRAMME LE PLUS SIMPLE.....	11
2. AVEC UNE DEUXIEME PAGE	11
3. AVEC UN « TEMPLATE ».....	12
4. AMELIORATIONS	13
a. Flask.....	13
b. Html	13
c. Essai http://127.0.0.1:5000/hello/	14
d. Essai http://127.0.0.1:5000/hello/titi	14
III. RECUPERATION D'UN FICHIER STATIQUE.....	14
1. AJOUT A LA FIN DU FICHIER FLASK	14
2. EXEMPLE DE FICHIER A RECUPERER.....	14
IV. UTILISATION DE TEMPLATE.....	15
1. POURQUOI UTILISER UN TEMPLATE ?.....	15
2. UTILISATION.....	15
3. AVEC DES FICHIERS CSS	16
V. AUTRES EXEMPLE D'ENVOI DE FICHIER STATIQUE	17
1. ENVOI DE FICHIER STATIQUE	17
2. TOUS LES FICHIERS UTILISES DANS CETTE PARTIE	18
a. Tous les fichiers : <i>app.py</i>	18
b. Tous les fichiers : <i>FruitsLegumes.json</i>	19
c. <i>Site.css</i>	19
d. <i>Bonjour.html</i>	19
e. <i>Settings.json</i>	20
VI. UTILISATION DES FRAGMENTS (SNIPPET)	20
1. PRINCIPE	20
2. CREATION D'UN SNIPPET	21
3. UTILISATION DE SNIPPER	22
a. <i>Home.html</i>	22
b. <i>apropos.html et contact</i>	23



4.	UN NOUVEAU CSS POUR LES NAVIGATIONS	23
5.	ET APP.PY	24
6.	LANCEMENT	24
7.	RESULTATS.....	27
8.	TOUS LES FICHIERS UTILISES DANS CETTE PARTIE	28
a.	App.py.....	28
b.	Launch.json	28
c.	Settings.json.....	29
d.	Layout.html dans template.....	29
e.	Home.html dans template	30
f.	Contact.html dans template	30
g.	Site.css dans static.....	30
h.	Html.json.....	31
VII.	UTILISATION D'UN FICHER DE DEPENDANCES	32



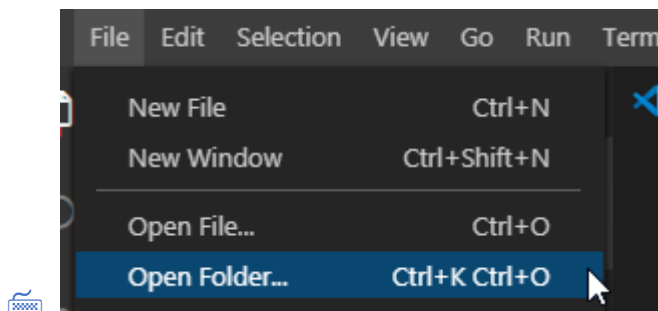
Ne pas oublier l'environnement virtuel

I. Créer un projet flask sous Visual Studio

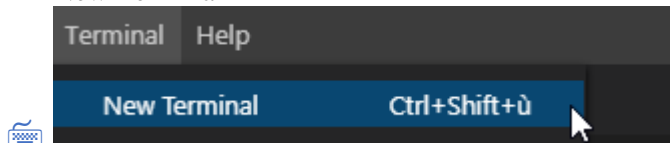
1. Configuration de l'environnement :

Créer un répertoire : D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet

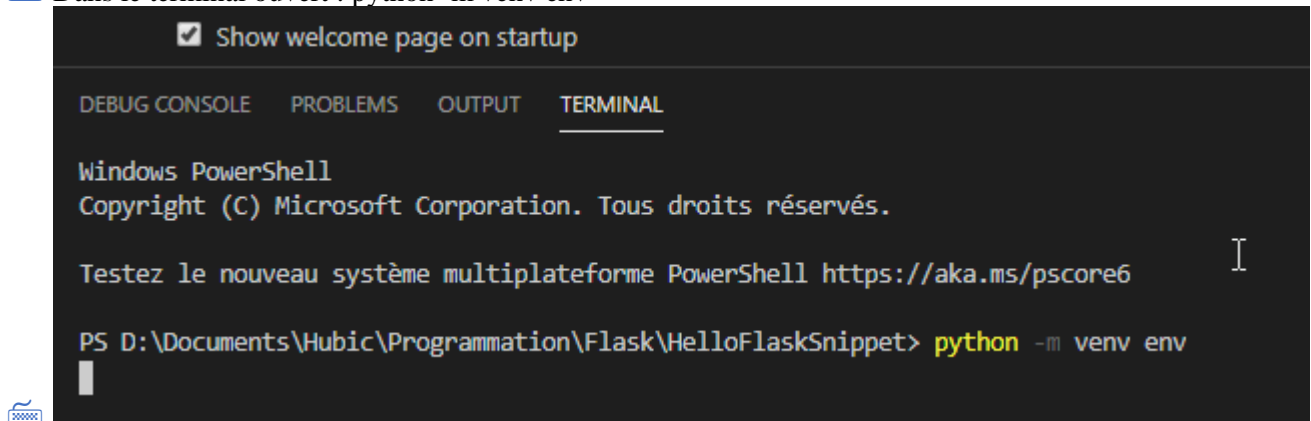
Dans VS : *Open Folder...*



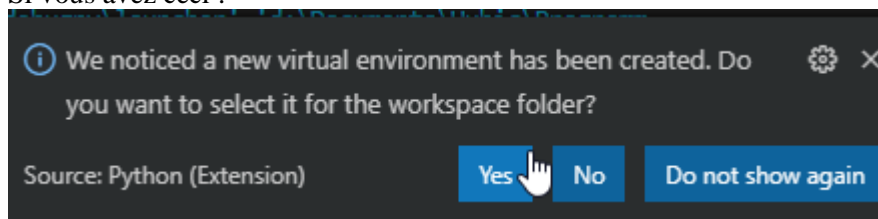
New Terminal



Dans le terminal ouvert : `python -m venv env`

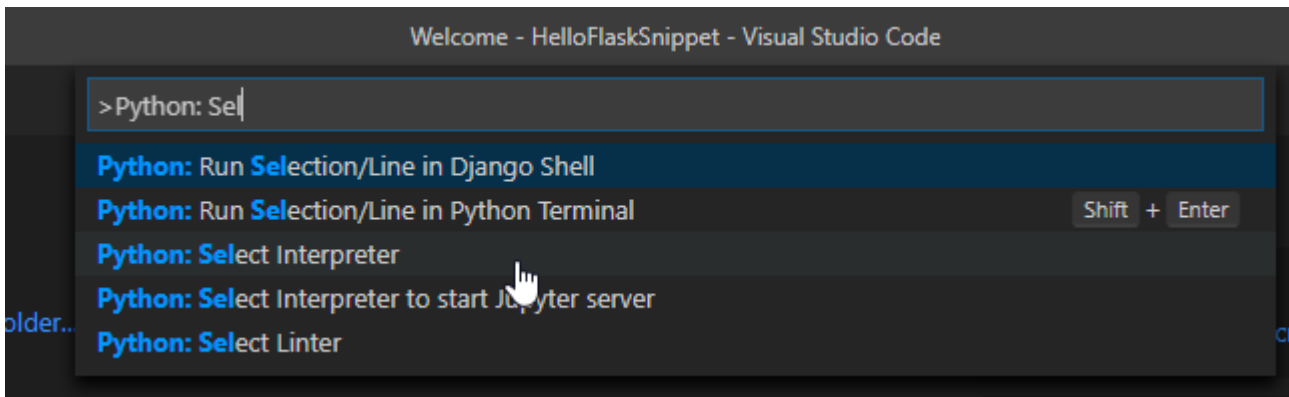


Si vous avez ceci :



Cliquez sur Yes

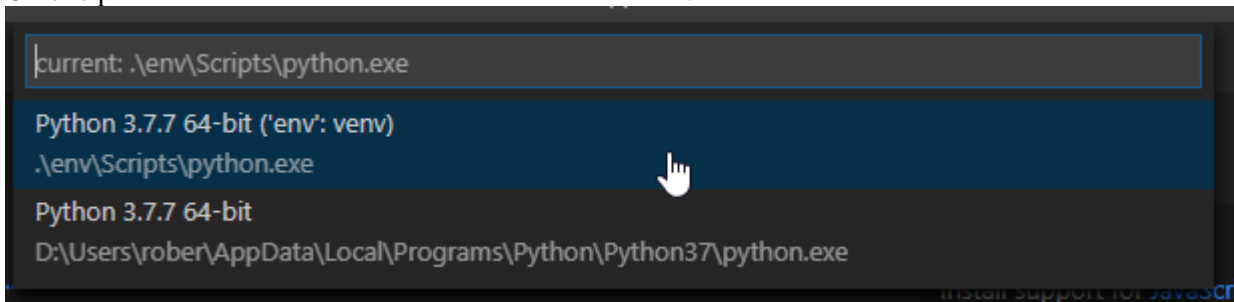
View > Command Palette et sélectionner **Python: Select Interpreter:**



Vérifier en bas à gauche :

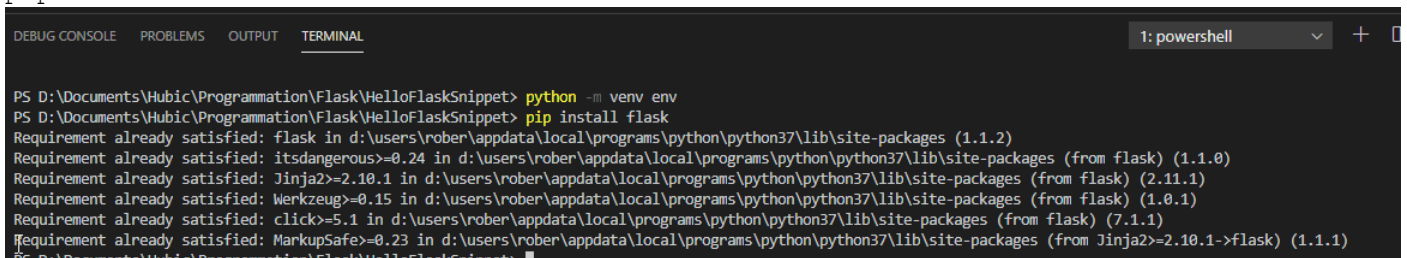


Un clic permet de modifier l'environnement si nécessaire :



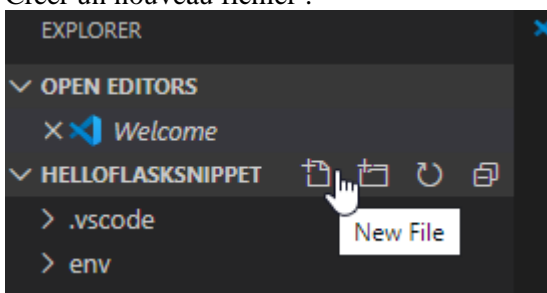
Dans le terminal, installer l'environnement :

```
pip install flask
```



2. Premier programme

Créer un nouveau fichier :



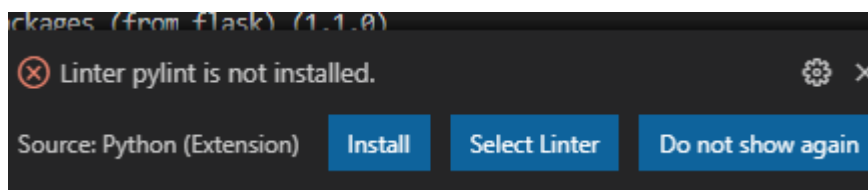
Appeler-le app.py et mettez ce code :



```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"
```

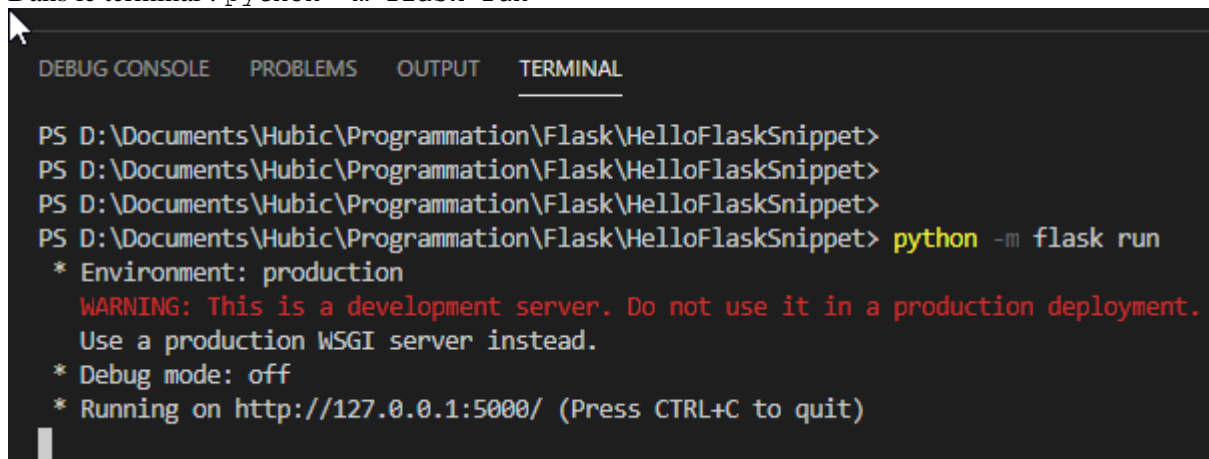
Il se peut qu'on vous demande d'installer Linter : ok



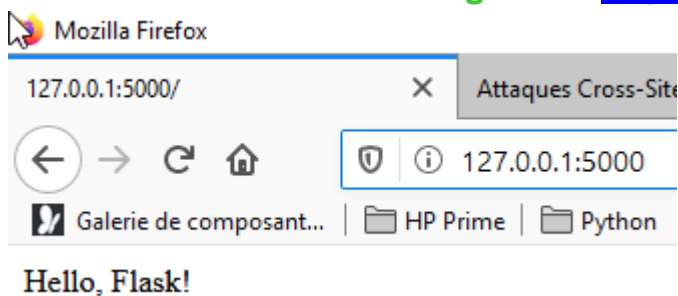
N'oubliez pas de sauvegarder

a. Lancement

Dans le terminal : `python -m flask run`



b. Dans un navigateur : <http://127.0.0.1:5000/>



3. Un autre code plus abouti

Pour terminer le code : CTRL C dans le terminal



a. Code flask

```
from flask import Flask
from datetime import datetime
import re

app = Flask(__name__)

@app.route("/") #127.0.0.1:5000
def home():
    return "Page d'accueil"

@app.route("/hello/") #127.0.0.1:5000/hello
def bonjour_sansNom():
    return bonjour(nom='Inconnu')

@app.route("/hello/<nom>") #127.0.0.1:5000/hello/unNom
def bonjour(nom='Inconnu'):
    now = datetime.now()
    formatted_now = now.strftime("%A, %d %B, %Y at %X")

    # Filter the name argument to letters only using regular expressions. URL arguments
    # can contain arbitrary text, so we restrict to safe characters only.
    match_object = re.match("[a-zA-Z]+", nom)

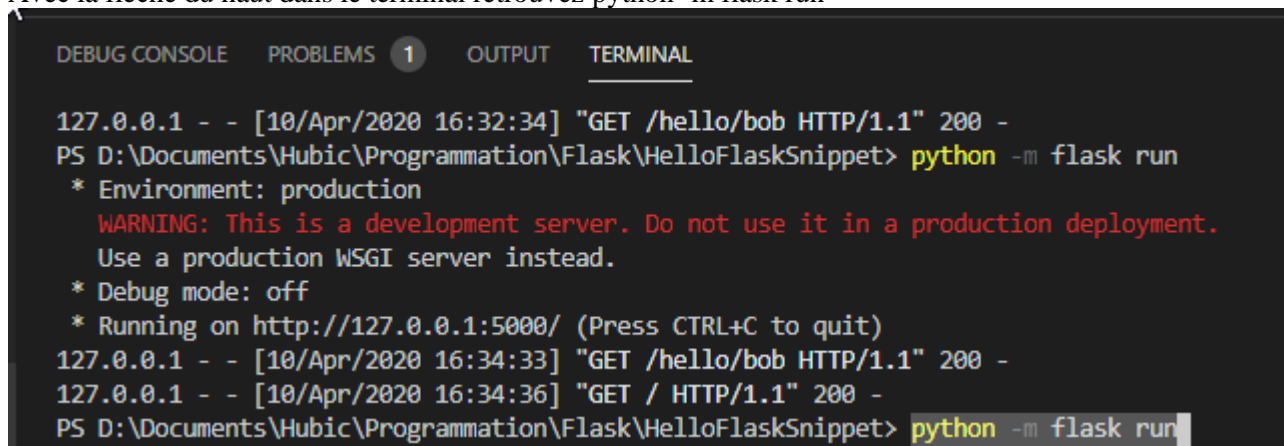
    if match_object:
        clean_name = match_object.group(0)
    else:
        clean_name = "Amie"

    content = "Bonjour " + clean_name + " il est exactement " + formatted_now
    return content
```

Pour lancer le programme :

CTRL C pour arrêter si ce n'est pas déjà fait

Avec la flèche du haut dans le terminal retrouvez `python -m flask run`

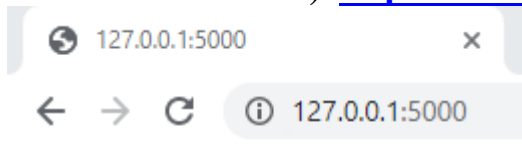


```
DEBUG CONSOLE PROBLEMS 1 OUTPUT TERMINAL
127.0.0.1 - - [10/Apr/2020 16:32:34] "GET /hello/bob HTTP/1.1" 200 -
PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> python -m flask run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [10/Apr/2020 16:34:33] "GET /hello/bob HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2020 16:34:36] "GET / HTTP/1.1" 200 -
PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> python -m flask run
```



b. Navigateur :

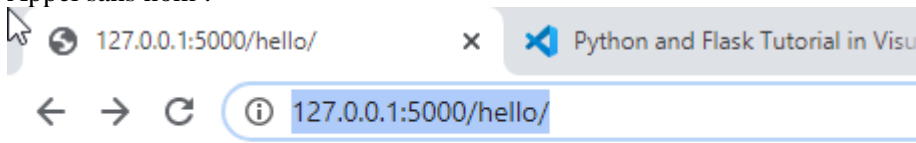
a) <http://127.0.0.1:5000/>



Page d'accueil

b) <http://127.0.0.1:5000/hello/>

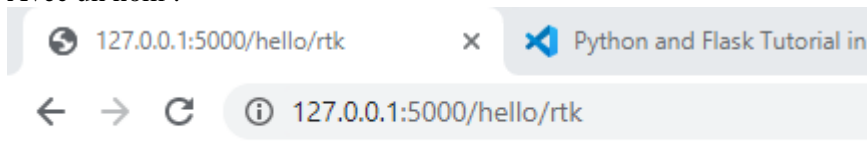
Appel sans nom :



Bonjour Inconnu il est exactement Friday, 10 April, 2020 at 16:44:59

c) <http://127.0.0.1:5000/hello/rtk>

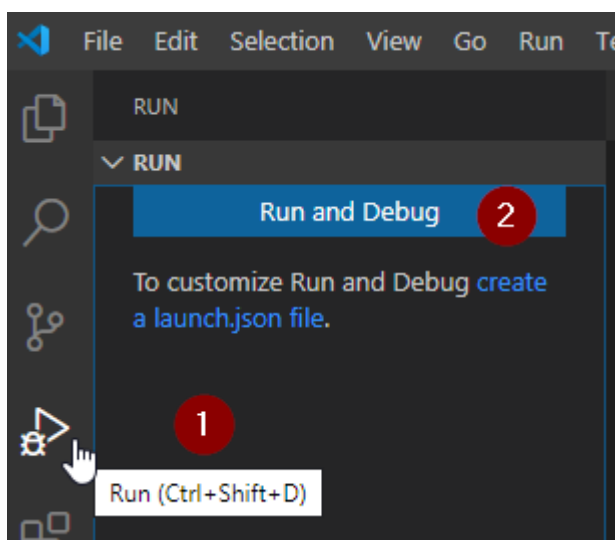
Avec un nom :



Bonjour rtk il est exactement Friday, 10 April, 2020 at 16:38:32

4. Avec un fichier de configuration c'est mieux

a. Création du fichier de lancement

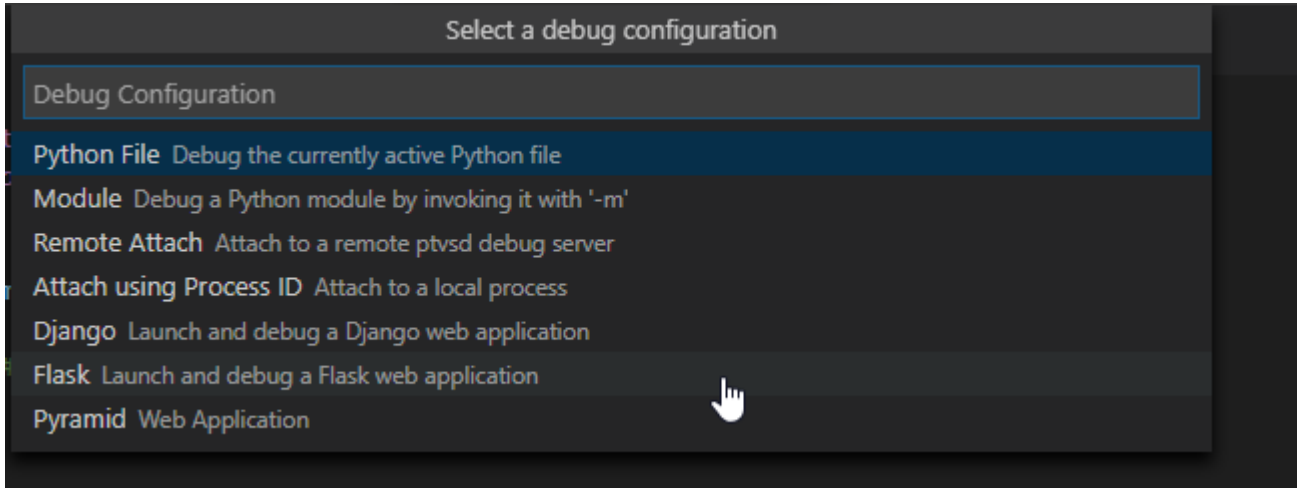


1. Débuggage



2. Création du fichier launch.json

Choisir Flask :



Puis vous avez accès au fichier launch.json :

```
Terminal Help launch.json - HelloFlaskSnippet - Visual Studio Code
Welcome app.py launch.json x
.vscode > {} launch.json > ...
1
2 // Use IntelliSense to learn about possible attributes.
3 // Hover to view descriptions of existing attributes.
4 // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5 "version": "0.2.0",
6 "configurations": [
7   {
8     "name": "Python : Flask",
9     "type": "python",
10    "request": "launch",
11    "module": "flask",
12    "env": {
13      "FLASK_APP": "app.py",
14      "FLASK_ENV": "development",
15      "FLASK_DEBUG": "0"
16    },
17    "args": [
18      "run",
19      "--no-debugger",
20      "--no-reload"
21    ],
22    "jinja": true
23  }
24 ]
25
```

Ensuite un bouton *Add Configuration* s'affiche :*Si nécessaire pour d'autres configurations PAS UTILE*



```
.vscode {} launch.json > ...
1
2 // Use IntelliSense to learn about possible attributes.
3 // Hover to view descriptions of existing attributes.
4 // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5 "version": "0.2.0",
6 "configurations": [
7   {
8     "name": "Python : Flask",
9     "type": "python",
10    "request": "launch",
11    "module": "flask",
12    "env": {
13      "FLASK_APP": "app.py",
14      "FLASK_ENV": "development",
15      "FLASK_DEBUG": "0"
16    },
17    "args": [
18      "run",
19      "--no-debugger",
20      "--no-reload"
21    ],
22    "jinja": true
23  }
24 ]
25
```

Add Configuration...

5. Débugger

a. Point d'arrêt

Mettez un point d'arrêt sur la ligne 13 par exemple

```
11 @app.route('/hello/<name>')
12 def hello_there(name):
13     now = datetime.now()
14     formatted_now = now.strftime("%A, %d %B, %Y at %X")
15
16     # Filter the name argument to letters only using regular
17     # can contain arbitrary text, so we restrict to safe char
18     match_object = re.match("[a-zA-Z]+", name)
19
```

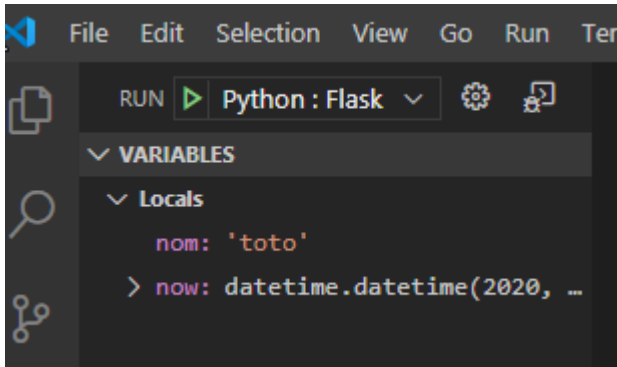
Lancez par F5 ou

Allez sur <http://127.0.0.1:5000/hello/toto>

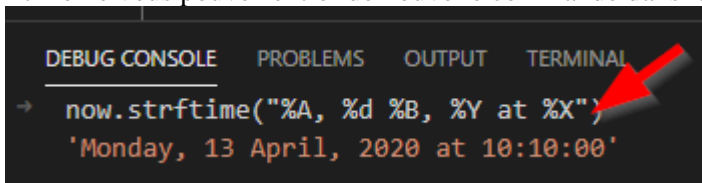
Le programme se met en pause :

```
16 def bonjour(nom='Inconnu'):
17     now = datetime.now()
18     formatted_now = now.strftime("%A, %d %B, %Y at %X")
19
```

Vous avez accès aux variables en haut à gauche :



Et même vous pouvez entrer de nouvelle commande dans le terminal de debug :



b. Erreur

Si lors du lancement vous avez le module *flask* n'est pas connue

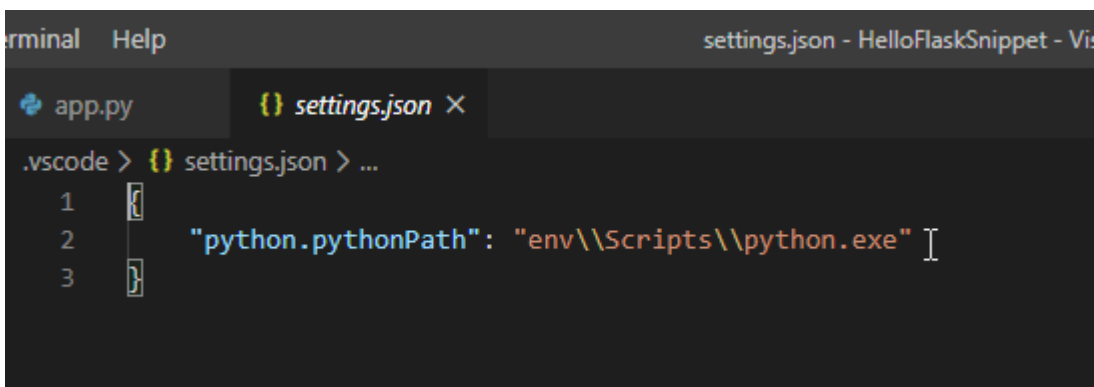
```
D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet\env\Scripts\python.exe: No module named flask
PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> █
```

Testez ceci :

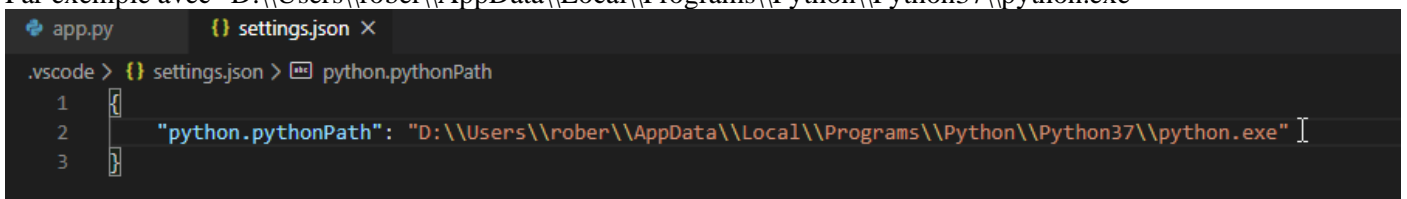
pip install ptvsd --upgrade dans le terminal :

```
PS D:\Documents\Hubic\Programmation\Flask\HelloFlask> pip install ptvsd --upgrade █
```

Réessayez sinon modifier le path de python dans le settings.json



Par exemple avec "D:\\Users\\rober\\AppData\\Local\\Programs\\Python\\Python37\\python.exe"





Et ensuite cela fonctionne :

```
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL 2: Python Debug Consc + [
Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> ${env:DEBUGPY_LAUNCHER_PORT}='56210'; & 'D:\Users\rober\AppData\Local\Programs\Python\Python37\python
.exe' 'c:\Users\rober\.vscode\extensions\ms-python.python-2020.3.71659\pythonFiles\lib\python\debugpy\wheels\debugpy\launcher' '-m' 'flask' 'run' '--no-debugger'
--no-reload'
* Serving Flask app "app.py"
* Environment: development
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> [
```

II. Premiers essais :

En suivant :

https://code.visualstudio.com/docs/python/tutorial-flask#_create-a-project-environment-for-the-flask-tutorial

<https://perso.liris.cnrs.fr/pierre-antoine.champin/2017/progweb-python/cours/cm2.html>

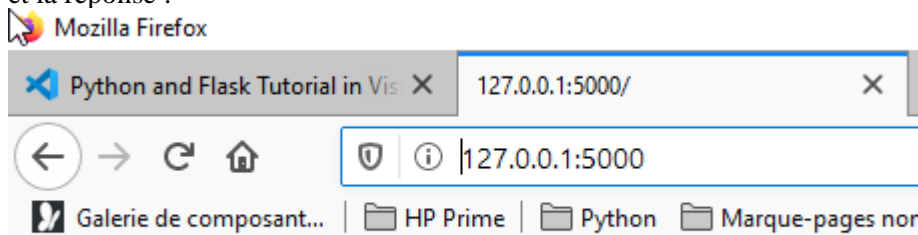
1. Programme le plus simple

```
from flask import Flask
from datetime import datetime
from flask import render_template
import re

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, Flask!"
```

et la réponse :



Hello, Flask!

2. Avec une deuxième page

```
@app.route("/hello/<name>")
def hello_there(name):
    print("http://127.0.0.1:5000/hello/bob")
    now = datetime.now()
    formatted_now = now.strftime("%A, %d %B, %Y at %X")

    # Filter the name argument to letters only using regular expressions. URL arguments
```



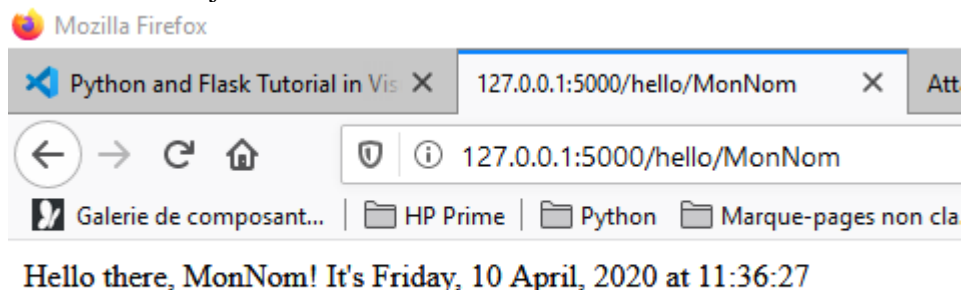
```
# can contain arbitrary text, so we restrict to safe characters only.
match_object = re.match("[a-zA-Z]+", name)

if match_object:
    clean_name = match_object.group(0)
else:
    clean_name = "Friend"

content = "Hello there, " + clean_name + "! It's " + formatted_now
return content
```

Dans le navigateur : <http://127.0.0.1:5000/hello/MonNom>

Attention aux majuscules !!!



3. Avec un « template »

Le template est hello_there.html :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello, Flask</title>
  </head>
  <body>
    {%if name %}
      <strong>Hello there, {{ name }}!</strong> It's {{ date.strftime("%A, %d %B, %Y
at %X") }}.
    {% else %}
      What's your name? Provide it after /hello/ in the URL.
    {% endif %}
  </body>
</html>
```

Le code flask :

```
from flask import Flask
from datetime import datetime
from flask import render_template
import re
```



```
app = Flask(__name__)
@app.route("/")
@app.route("/hello/")
@app.route("/hello/<name>")
def hello_there(name = "rtk"):
    return render_template(
        "hello_there.html",
        name=name,
        date=datetime.now()
    )
```

- L'appel par <http://127.0.0.1:5000/hello/toto>
Hello there, toto! It's Friday, 10 April, 2020 at 11:41:21. Zz
- Et les appels par <http://127.0.0.1:5000/hello/> ou <http://127.0.0.1:5000/>
Hello there, rtk! It's Friday, 10 April, 2020 at 11:42:17.

4. Améliorations

a. Flask

```
from flask import Flask
from datetime import datetime
from flask import render_template
import re

app = Flask(__name__)
@app.route("/")
@app.route("/hello/")
def hello_there_sansNom(): # Sans nom
    return render_template(
        "hello_there-03.html",
        name=None,
        date=datetime.now()
    )
@app.route("/hello/<name>")
def hello_there(name = "rtk"): #avec un nom par défaut (ça ne sert à rien ici)
    return render_template(
        "hello_there-03.html",
        name=name,
        date=datetime.now()
    )
```

b. Html

```
<!DOCTYPE html>
```



```
<HEAD>
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.css')}}"
/>
</HEAD>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello, Flask</title>
  </head>
  <body>
    {%if name %}
      <span class="message">Bonjour, {{ name }}!</span> il est {{ date.strftime("%A,
%d %B, %Y at %X") }}.
    {% else %}
      <span class="message">Quel est votre nom ? Fournissez-
le juste après /hello/ dans l'url.</span>
    {% endif %}
  </body>
</html>
```

c. Essai <http://127.0.0.1:5000/hello/>

Quel est votre nom ? Fournissez-le juste après /hello/ dans l'url.

d. Essai <http://127.0.0.1:5000/hello/titi>

Bonjour, titi! il est Friday, 10 April, 2020 at 11:58:32.

III. Récupération d'un fichier statique

Dans cet exemple c'est un fichier json

1. Ajout à la fin du fichier flask

```
@app.route("/fichier/data")
def get_data():
    return app.send_static_file("data.json")
```

2. Exemple de fichier à récupérer

```
{
  "01": {
    "note": "Exemple du principe de fonctionnement."
  }
}
```



IV. Utilisation de template

1. Pourquoi utiliser un template ?

Même si il est possible de générer directement du html, il est préférable de ne pas le faire afin de prévenir des attaques de XSS.

2. Utilisation

- 📁 Créez un repertoire *templates*
- 📄 Y mettre ce fichier appelé *bonjour.html*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello, Flask</title>
  </head>
  <body>
    {%if name %}
      <strong>Bonjour {{ name }}!</strong> il est {{ date.strftime("%A, %d %B, %Y a
t %X") }}.
    {% else %}
      Quel est votre nom ? Vous pouvez le fournir à la suite de /hello/ dans l'URL.
    {% endif %}
  </body>
</html>
```

Remarques :

- Ce template possède deux variables : name et date
- Notez leur gestion entre {}

- 📄 Modifiez le code app.py ainsi :

```
from flask import Flask
from datetime import datetime
import re
from flask import render_template # pour utiliser les templates

app = Flask(__name__)

@app.route("/") #127.0.0.1:5000 toujours le même message
def home():
    return "Page d'accueil"

@app.route("/hello/") #127.0.0.1:5000/hello sans nom
@app.route("/hello/<nom>")#127.0.0.1:5000/hello/ suivi d'un nom
def bonjour(nom = None):
    return render_template("bonjour.html",
```




```
name = nom,
date=datetime.now()
)
```

Les lignes importantes sont :

```
from flask import render_template # pour utiliser les templates
importe render template qui va nous servir à utiliser les templates
```

```
return render_template("bonjour.html",
name = nom,
date=datetime.now()
)
```

Appel du template avec deux variables : name et date

3. Avec des fichiers css

1. Par défaut *flask* il est possible de stocker des ressources statiques comme des images et du css dans un répertoire qui se nomme *static*. Créez ce répertoire *static*
2. A l'intérieur ce fichier *site.css*

```
.message {
font-weight: 600;
color: blue;
}
```

Ce fichier va créer une classe css dont la police est 600 et la couleur bleue

3. Modifiez *bonjour.html* ainsi :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello, Flask</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename=
'site.css')}}" />
  </head>
  <body>
    {%if name %}
      <span class="message">Bonjour {{ name }} !</span> il est {{ date.strftime(
"%A, %d %B, %Y at %X") }}.
    {% else %}
      <span class="message">Quel est votre nom ? </span> Vous pouvez le fournir à la
suite de /hello/ dans l'URL.
    {% endif %}
  </body>
</html>
```

Remarques :



1. Ce fichier va appeler le css du site

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.css') }}" />
```

2. Et va afficher les messages à l'aide de la classe message

```
<span class="message">Quel est votre nom ? </span>
```

3. L'URL des fichiers statique sont données par la fonction url_for :

```
url_for('static', filename='site.css')
```

V. Autres exemple d'envoi de fichier statique

1. Envoi de fichier statique

Créez un fichier *FruitsLegumes.json* dans le répertoire *static* :

```
{  
  "fruits": [  
    { "kiwis": 3,  
      "mangues": 4,  
      "pommes": null  
    },  
    { "panier": true }  
  ],  
  "legumes": {  
    "patates": "amandine",  
    "poireaux": false  
  },  
  "viandes": ["poisson", "poulet", "boeuf"]  
}
```

Ajoutez dans *app.py* :

```
@app.route("/fruits")  
def get_fruitslegumes():  
    return app.send_static_file("FruitsLegumes.json")
```

testez

<http://127.0.0.1:5000/fruits>



← → ↻ ⓘ 127.0.0.1:5000/fruits

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true }
  ],
  "legumes": {
    "patates": "amandine",
    "poireaux": false
  },
  "viandes": ["poisson", "poulet", "boeuf"]
}
```

2. Tous les fichiers utilisés dans cette partie

app.py × {} FruitsLegumes.json # site.css <> bonjour.html {} settings.json

a. Tous les fichiers : app.py

```
from flask import Flask
from datetime import datetime
import re
from flask import render_template # pour utiliser les templates

app = Flask(__name__)

@app.route("/") #127.0.0.1:5000 toujours le même message
def home():
    return "Page d'accueil"

@app.route("/hello/") #127.0.0.1:5000/hello sans nom
@app.route("/hello/<nom>") #127.0.0.1:5000/hello/ suivi d'un nom
def bonjour(nom = None):
    return render_template("bonjour.html",
        name = nom,
        date=datetime.now()
    )

@app.route("/fruits")
def get_fruitslegumes():
    return app.send_static_file("FruitsLegumes.json")
```



b. Tous les fichiers : FruitsLegumes.json

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true }
  ],
  "legumes": {
    "patates": "amandine",
    "poireaux": false
  },
  "viandes": ["poisson", "poulet", "boeuf"]
}
```

c. Site.css

```
.message {
  font-weight: 600;
  color: blue;
}
```

d. Bonjour.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Hello, Flask</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.css')}}"/>
  </head>
  <body>
    {%if name %}
      <span class="message">Bonjour {{ name }} !</span> il est {{ date.strftime("%A, %d %B, %Y at %X") }}.
    {% else %}
      <span class="message">Quel est votre nom ? </span> Vous pouvez le fournir à la suite de /hello/ dans l'URL.
    {% endif %}
  </body>
</html>
```



e. Settings.json

```
app.py  FruitsLegumes.json  # site.css  < > bonjour.html  {} settings.json X
.vscode > {} settings.json > ...
1  {
2  .. "python.pythonPath": "D:\\Users\\rober\\AppData\\Local\\Programs\\Python\\Python37\\python.exe"
3  }
```

VI. Utilisation des fragments (snippet)

1. Principe

App.py va appeler les fichiers `home.html`, `apropos.html` et `contact.html` qui ressemble à (ex de `contact.html`)

```
{% extends "layout.html" %}
{% block title %}
Contact
{% endblock %}
{% block content %}
<span class="message">Page de contact</span>
{% endblock %}
```

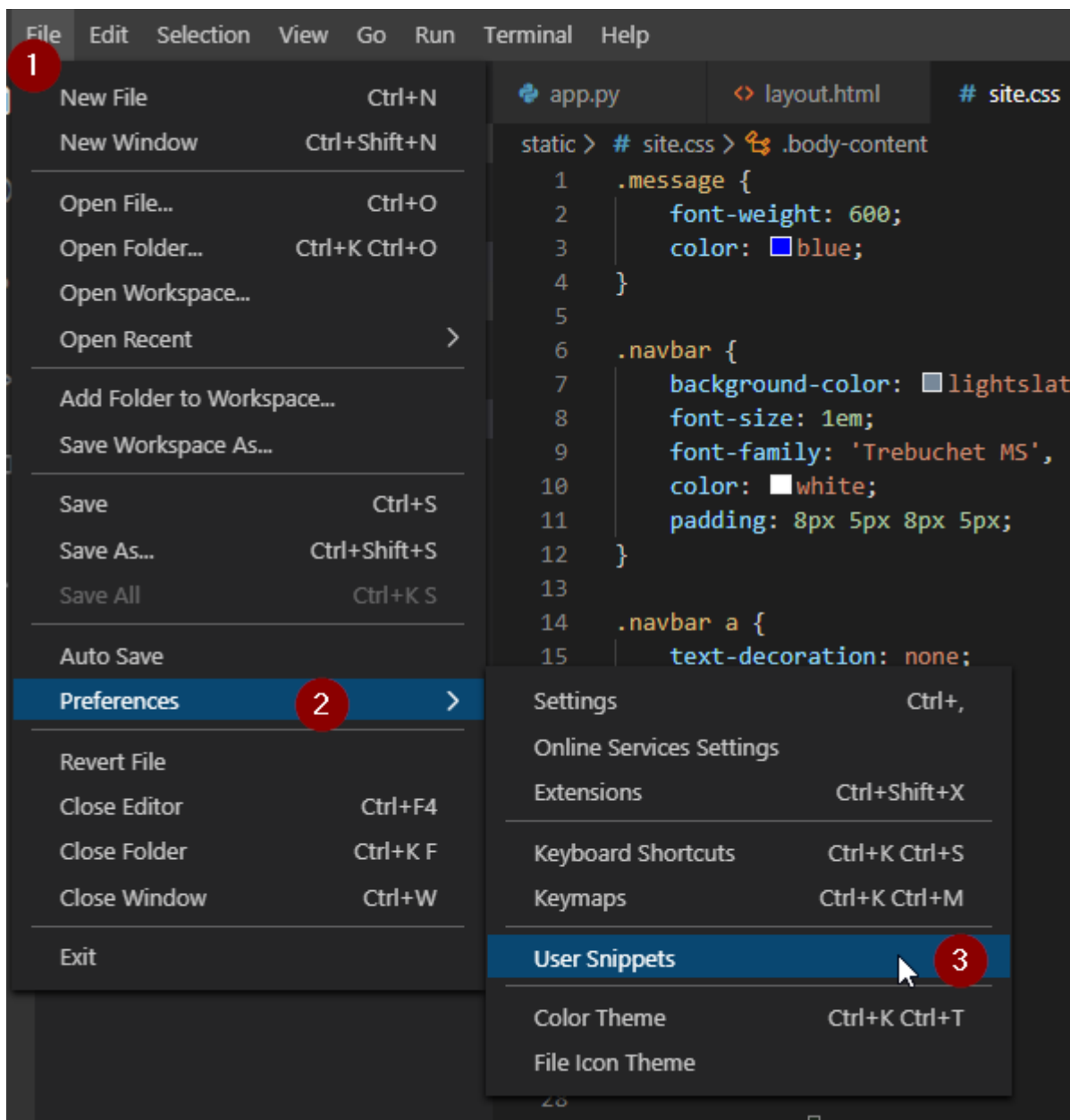
Ces trois fichiers vont utiliser un template `layout.html` qui va imposer la structure :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.c
ss')}}" />
  </head>
  <body>
    <div class="navbar">
      <a href="{{ url_for('home') }}" class="navbar-brand">Home</a>
      <a href="{{ url_for('about') }}" class="navbar-item">About</a>
      <a href="{{ url_for('contact') }}" class="navbar-item">Contact</a>
    </div>
    <div class="body-content">
      {% block content %}
      {% endblock %}
    </div>
    <hr/>
    <footer>
      <p>Essai rtk 2020</p>
    </footer>
```



```
</div>  
</body>  
</html>
```

2. Création d'un snippet



Puis :





Un fichier `html.json` s'ouvre automatiquement :

`{}` `html.json` ✕

Y mettre avant l'accolade de la `}` de la fin :

```
"Flask Tutorial: template extending layout.html": {
  "prefix": "flexlayout",
  "body": [
    "{% extends \"layout.html\" %}",
    "{% block title %}",
    "$0",
    "{% endblock %}",
    "{% block content %}",
    "{% endblock %}"
  ],
  "description": "Boilerplate template that extends layout.html"
},
```

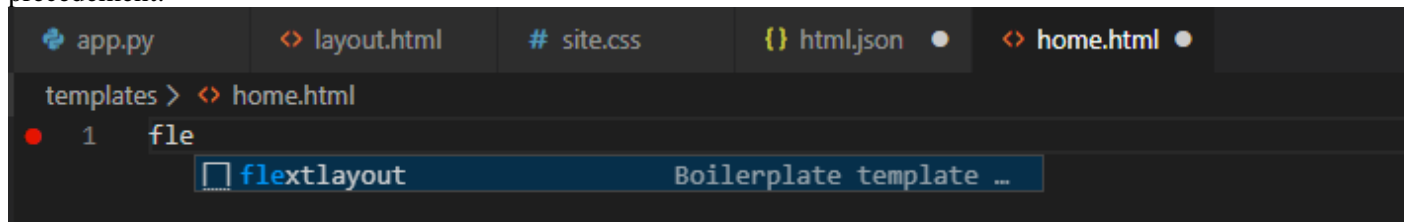
Sauvegardez tous vos fichiers.

3. Utilisation de snippet

a. Home.html

Créez un fichier `home.html` dans le répertoire `templates`.

Lorsque vous entrez les lettres `flex`, VS vous propose un des snippets disponibles, dans notre exemple `flexlayout` créé précédemment:



Cliquez dessus et voilà :

```
{% extends "layout.html" %}
{% block title %}

{% endblock %}
{% block content %}
{% endblock %}
```

Modifiez ce fichier en ajoutant

Home

Et

```
<p>Home page : exemple de snippet </p>
```

Ainsi :



Flask avec Visual Studio

```
{% extends "layout.html" %}
{% block title %}
Home
{% endblock %}
{% block content %}
<p>Home page : exemple de snippet </p>
{% endblock %}
```

b. apropos.html et contact

 Faites de même pour apropos.html et contact mais cette fois en utilisant une classe css :

Apropos.html :

```
{% extends "layout.html" %}
{% block title %}
A propos
{% endblock %}
{% block content %}
<span class="message">Page : à propos de snippet et d'autres</span>
{% endblock %}
```

Contact.html

```
{% extends "layout.html" %}
{% block title %}
Contact
{% endblock %}
{% block content %}
<span class="message">Page de contact</span>
{% endblock %}
```

4. Un nouveau css pour les navigations

```
.message {
  font-weight: 600;
  color: blue;
}

.navbar {
  background-color: lightslategray;
  font-size: 1em;
  font-
family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans
-serif;
  color: white;
  padding: 8px 5px 8px 5px;
}

.navbar a {
  text-decoration: none;
```



```
color: inherit;
}

.navbar-brand {
  font-size: 1.2em;
  font-weight: 600;
}

.navbar-item {
  font-variant: small-caps;
  margin-left: 30px;
}

.body-content {
  padding: 5px;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
```

5. Et app.py

```
from flask import Flask
from datetime import datetime
import re
from flask import render_template # pour utiliser les templates

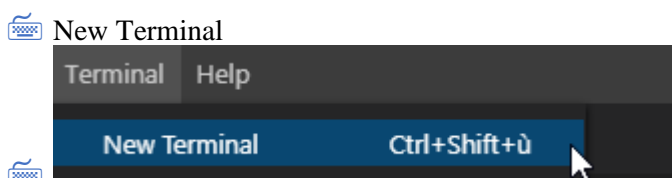
app = Flask(__name__)

@app.route("/") #127.0.0.1:5000 toujours le même message
def home():
    return render_template("home.html")

# New functions
@app.route("/about/")
def about():
    return render_template("apropos.html")

@app.route("/contact/")
def contact():
    return render_template("contact.html")
```

6. Lancement

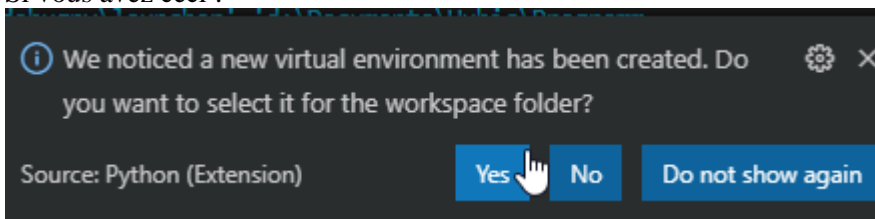




Dans le terminal ouvert : `python -m venv env`

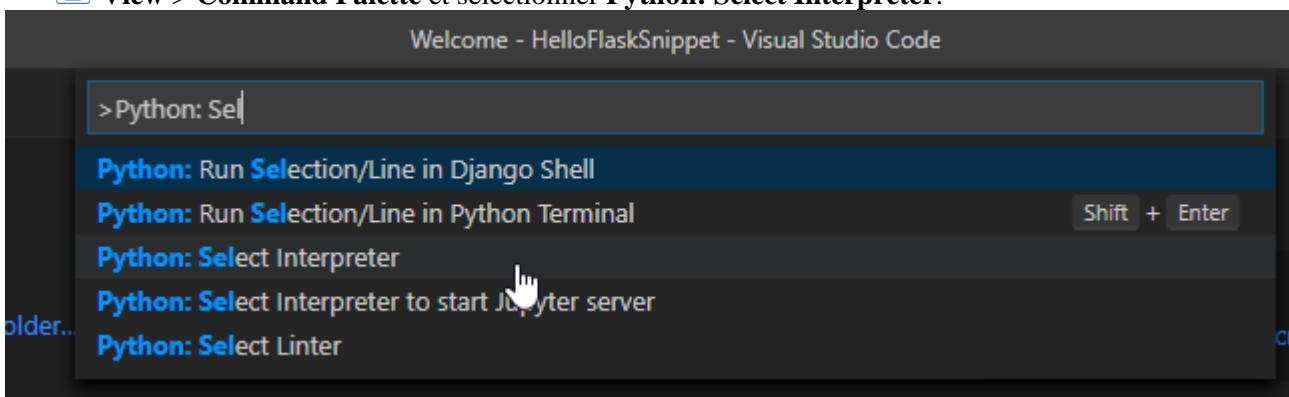
```
☑ Show welcome page on startup  
DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL  
Windows PowerShell  
Copyright (C) Microsoft Corporation. Tous droits réservés.  
Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6  
PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> python -m venv env
```

Si vous avez ceci :

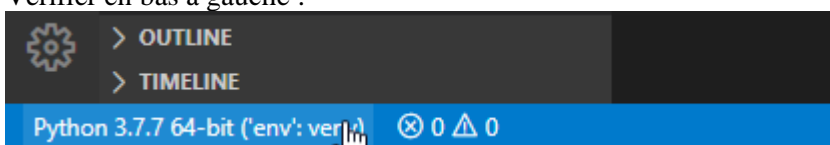


Cliquez sur Yes

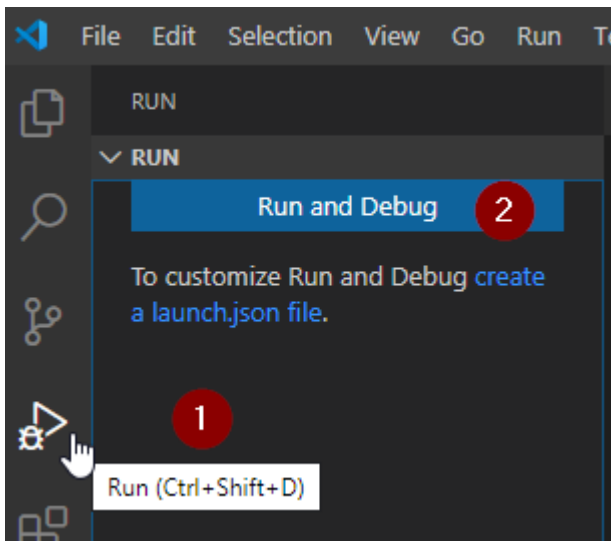
View > Command Palette et sélectionner **Python: Select Interpreter:**



Vérifier en bas à gauche :

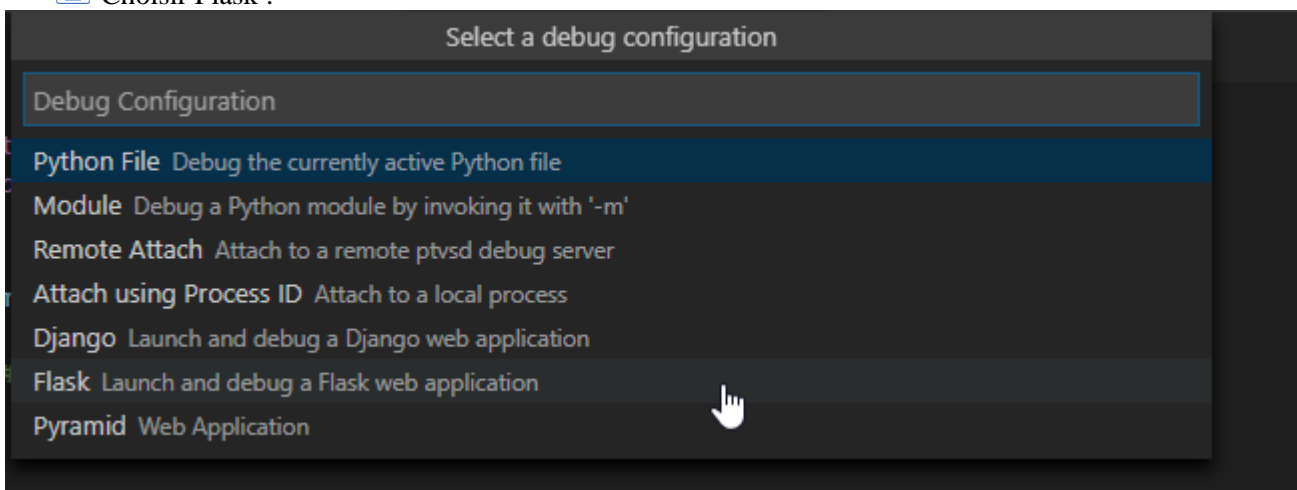


Création du fichier de lancement



3. Débuggage
4. Création du fichier launch.json

Choisir Flask :



Puis vous avez accès au fichier launch.json

Si lors du lancement vous avez le module *flask* n'est pas connue

```
D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet\env\Scripts\python.exe: No module named flask  
PS D:\Documents\Hubic\Programmation\Flask\HelloFlaskSnippet> █
```

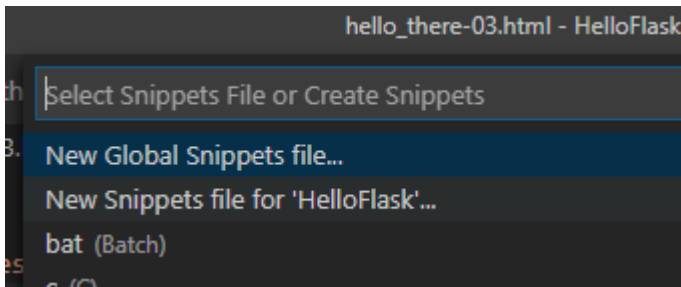
Modifiez *settings.json* en y mettant le chemin complet de votre interpréteur python :

```
{  
  "python.pythonPath": "D:\\Users\\rober\\AppData\\Local\\Programs\\Python\\Python37\\python.exe"  
}
```



7. Résultats

http://127.0.0.1:5000/	http://127.0.0.1:5000/about/	http://127.0.0.1:5000/contact/
Home About Contact Home page : exemple de snippet ----- Essai rtk 2020	Home About Contact Page : à propos de snippet et d'autres ----- Essai rtk 2020	Home About Contact Page de contact ----- Essai rtk 2020



8. Tous les fichiers utilisés dans cette partie



a. App.py

```
from flask import Flask
from datetime import datetime
import re
from flask import render_template # pour utiliser les templates

app = Flask(__name__)

@app.route("/") #127.0.0.1:5000 toujours le même message
def home():
    return render_template("home.html")

# New functions
@app.route("/about/")
def about():
    return render_template("apropos.html")

@app.route("/contact/")
def contact():
    return render_template("contact.html")
```

b. Launch.json

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python : Flask",
      "type": "python",
```



```
"request": "launch",
"module": "flask",
"env": {
  "FLASK_APP": "app.py",
  "FLASK_ENV": "development",
  "FLASK_DEBUG": "0"
},
"args": [
  "run",
  "--no-debugger",
  "--no-reload"
],
"jinja": true
}
]
```

c. Settings.json

```
{
  "python.pythonPath": "D:\\Users\\rober\\AppData\\Local\\Programs\\Python\\Python37\\python.exe"
}
```

d. Layout.html dans *template*

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='site.css')}}" />
  </head>
  <body>
    <div class="navbar">
      <a href="{{ url_for('home') }}" class="navbar-brand">Home</a>
      <a href="{{ url_for('about') }}" class="navbar-item">About</a>
      <a href="{{ url_for('contact') }}" class="navbar-item">Contact</a>
    </div>
    <div class="body-content">
      {% block content %}
      {% endblock %}
    </div>
    <hr/>
    <footer>
      <p>Essai rtk 2020</p>
    </footer>
  </body>
</html>
```




```
</footer>  
</div>  
</body>  
</html>
```

e. Home.html dans *template*

```
{% extends "layout.html" %}  
{% block title %}  
Home  
{% endblock %}  
{% block content %}  
<p>Home page : exemple de snippet </p>  
{% endblock %}
```

f. Contact.html dans *template*

```
{% extends "layout.html" %}  
{% block title %}  
Contact  
{% endblock %}  
{% block content %}  
<span class="message">Page de contact</span>  
{% endblock %}
```

g. Site.css dans *static*

```
.message {  
  font-weight: 600;  
  color: blue;  
}  
  
.navbar {  
  background-color: lightslategray;  
  font-size: 1em;  
  font-  
family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans  
-serif;  
  color: white;  
  padding: 8px 5px 8px 5px;  
}  
  
.navbar a {  
  text-decoration: none;  
  color: inherit;  
}  
  
.navbar-brand {
```



Flask avec Visual Studio

```
font-size: 1.2em;
font-weight: 600;
}

.navbar-item {
  font-variant: small-caps;
  margin-left: 30px;
}

.body-content {
  padding: 5px;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
```

h. Html.json


```
{
  // Place your snippets for html here. Each snippet is defined under a snippet name and
  // has a prefix, body and
  // description. The prefix is what is used to trigger the snippet and the body will be
  // expanded and inserted. Possible variables are:
  // $1, $2 for tab stops, $0 for the final cursor position, and ${1:label}, ${2:another
  // } for placeholders. Placeholders with the
  // same ids are connected.
  // Example:
  // "Print to console": {
  //   "prefix": "log",
  //   "body": [
  //     "console.log('$1');",
  //     "$2"
  //   ],
  //   "description": "Log output to console"
  // }
  "Flask Tutorial: template extending layout.html": {
    "prefix": "flexlayout",
    "body": [
      "{% extends \"layout.html\" %}",
      "{% block title %}",
      "$0",
      "{% endblock %}",
      "{% block content %}",
      "{% endblock %}"
    ],
    "description": "Boilerplate template that extends layout.html"
  },
}
```




VII. Utilisation d'un fichier de dépendances

Il est souvent utile de récupérer la liste des différentes bibliothèques utilisées dans un ancien projet pour l'utiliser dans un nouveau.

Par convention un fichier nommé requirements.txt établit une telle liste.

 Dans le précédent projet, entrez dans le shell windows :

```
pip list > requirements.txt
```

 Dans le nouveau :

```
pip install -r requirements.txt
```