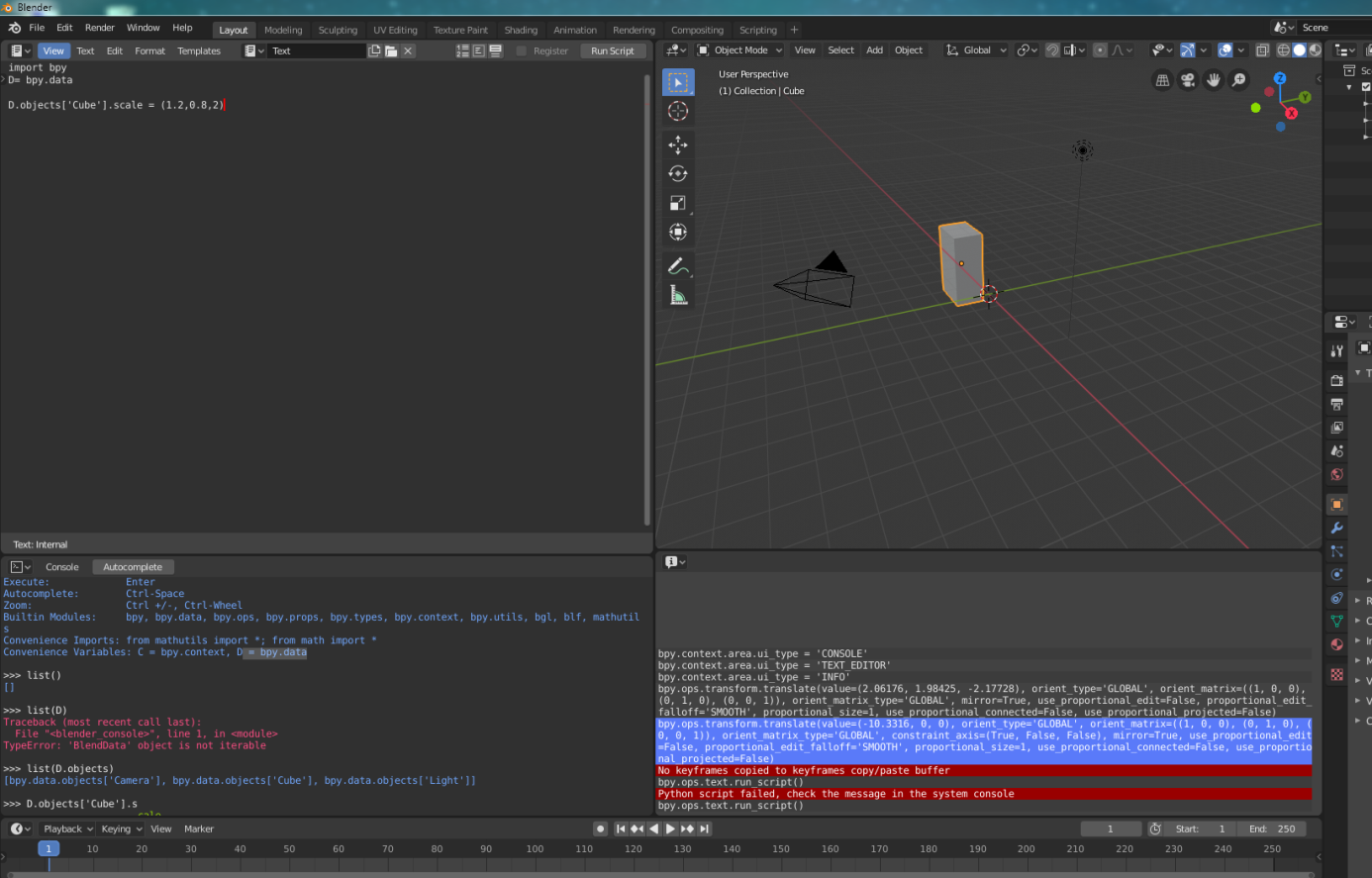
**Intro au scripting sous Blender :**

On peut se préparer un environnement de travail immédiat avec Python et Blender en positionnant une fenêtre en TextEditor, une fenêtre en Python console ,la vue 3D Viewport et une fenêtre avec les infos. On peut alors exécuter des commandes en 3D, on verra la commande script s’afficher en info, on peut exécuter des commandes directes dans la console qui importe déjà tout les modules bpy, ou on peut utiliser l’éditeur de teste en important explicitement bpy et avec le Run Script..



**Addon Blender :**

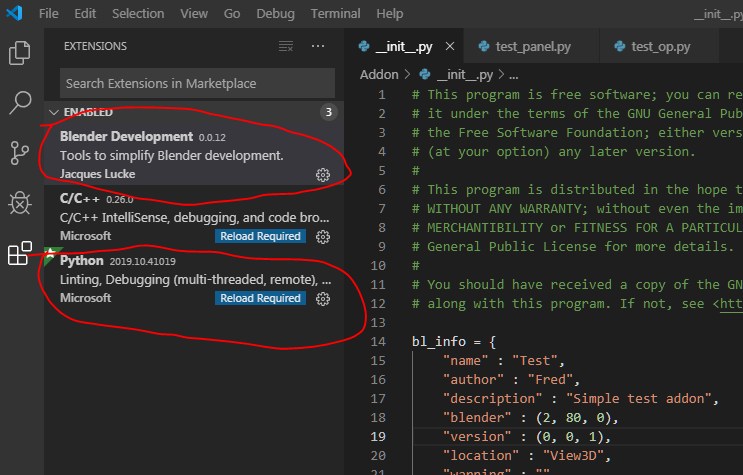
Installation environnement avec Blender 2.8 et Visual Studio Code comme éditeur de projet ( avec l’extension Blender qui va bien )

<https://www.youtube.com/watch?v=q06-hER7Y1Q>

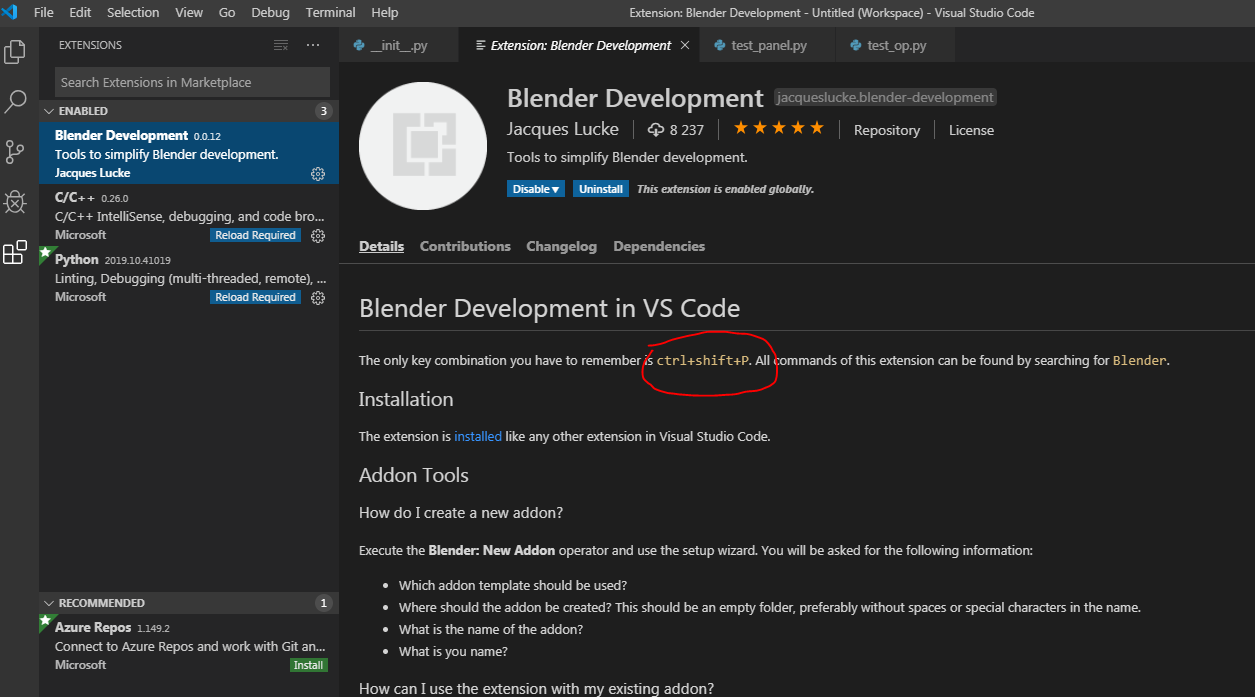
Exemple d’ajout d’addon sous forme de projet avec plusieurs fichiers Python :

<https://www.youtube.com/watch?v=uahfuypQQ04>

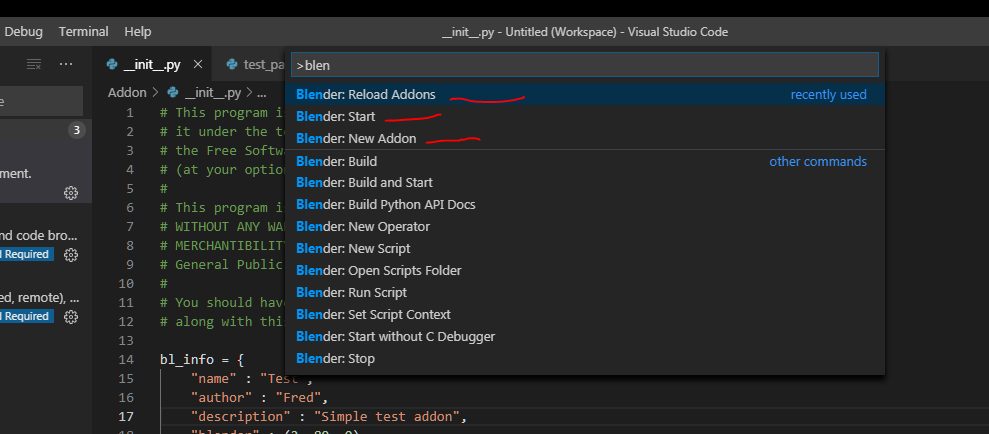
Visual Studio Code : Installation des AddOn :



Action à retenir : Ctrl+Shift+P



Avec Ctrl + Shift + P  on a la liste des actions à faire suivant les addons :



Exemple de AddON :

<https://docs.blender.org/manual/fr/dev/advanced/scripting/addon_tutorial.html#id1>

<https://www.youtube.com/watch?v=KNa5kJd2Epo>

<https://www.youtube.com/watch?v=mH0O6-XBoys>

<https://www.youtube.com/watch?v=ykcHggrUW_c>

<https://www.youtube.com/watch?v=K0yb4sZ7B4g>

Créer un nouveau Addon avec Visual Studio Code :

|  |  |
| --- | --- |
|  | Pour démarrer avec un environnement vide : Close Folder ou close WorkSpace |
|  | **IMPORTANT :**  Pour créer un nouveau ‘projet’, on va préparer un nouveau répertoire : ICI AddonTestA.  **Les addons et l’extension Blender travaillent sur les répertoires donc 1 addon = 1 répertoire.** |
|  | Ctrl Shift P  Puis Blender : New Addon |
|  | Simple pour le moment |
|  | Nom pour l’addon  Puis nom de l’auteur. |
|  | Puis demande du dossier où mettre les fichiers. **C’est le Folder qu’on devra ouvrir au besoin dans VScode.** |
|  | Creation auto de l’addon avec le fichier \_\_init\_\_.py dans le repertoire spécifié. |
|  |  |

Un petit tour du côté des extensions VSCode et voir comment marche l’extension pour Blender.. (quand ça marche pas justement )…:

|  |  |
| --- | --- |
|  | Les extensions rajoutées à VSCode |
|  | L’extension pour Blender :  On remarque par exemple que lors du lancement de Blender via l’extension de VSCode on a ( vu dans le terminal ) la commande :  *c:\Program Files\Blender Foundation\Blender\blender.exe --python c:\Users\Fauvel\.vscode\extensions\jacqueslucke.blender-development-0.0.12\pythonFiles\launch.py*  Qui s’exécute.. On peut aller voir le code .. surtout si le terminal donne des problèmes d’exécutions.. |
| **EXTRAIT DU TERMINAL SUR LANCMENT DE BLENDER après avoir bidouillé dans nos Addons et avoir fait un truc a priori qui n’est pas permis.. :**  > Executing task: c:\Program Files\Blender Foundation\Blender\blender.exe --python c:\Users\Fauvel\.vscode\extensions\jacqueslucke.blender-development-0.0.12\pythonFiles\launch.py <  Read prefs: C:\Users\Fauvel\AppData\Roaming\Blender Foundation\Blender\2.80\config\userpref.blend  found bundled python: c:\Program Files\Blender Foundation\Blender\2.80\python  Traceback (most recent call last):  File "c:\Program Files\Blender Foundation\Blender\2.80\scripts\modules\addon\_utils.py", line 351, in enable  mod = \_\_import\_\_(module\_name)  **ModuleNotFoundError: No module named 'Addon'**  [{'load\_dir': 'd:\\Etude\\Blender\\AddonsTests\\Addon', 'module\_name': 'Addon'}]  **HELLO pour essai de debug FFL**  **Debug : Source path : d:\Etude\Blender\AddonsTests\Addon**  **Debug : directory : d:\Etude\Blender\AddonsTests\Addon**  **Debug link\_path : : C:\Users\Fauvel\AppData\Roaming\Blender Foundation\Blender\2.80\scripts\addons\Addon**  Traceback (most recent call last):  File "c:\Users\Fauvel\.vscode\extensions\jacqueslucke.blender-development-0.0.12\pythonFiles\launch.py", line 19, in <module>  allow\_modify\_external\_python=os.environ['ALLOW\_MODIFY\_EXTERNAL\_PYTHON'] == "yes",  File "c:\Users\Fauvel\.vscode\extensions\jacqueslucke.blender-development-0.0.12\pythonFiles\include\blender\_vscode\\_\_init\_\_.py", line 14, in startup  path\_mappings = load\_addons.setup\_addon\_links(addons\_to\_load)  File "c:\Users\Fauvel\.vscode\extensions\jacqueslucke.blender-development-0.0.12\pythonFiles\include\blender\_vscode\load\_addons.py", line 22, in setup\_addon\_links  create\_link\_in\_user\_addon\_directory(source\_path, load\_path)  File "c:\Users\Fauvel\.vscode\extensions\jacqueslucke.blender-development-0.0.12\pythonFiles\include\blender\_vscode\load\_addons.py", line 47, in create\_link\_in\_user\_addon\_directory  \_winapi.CreateJunction(str(directory), str(link\_path))  **FileExistsError: [WinError 183] Impossible de cr├®er un fichier d├®j├á existant** | Lancement de la commande pour l’ouverture de Blender en passant des paramètres en ligne de commande.  On a des erreurs  Print() rajouté dans le code de l’extension pour voir ou ça se passe  Ici le path qui nous interesse est ici :  **C:\Users\Fauvel\AppData\Roaming\Blender Foundation\Blender\2.80\scripts\addons\Addon**  Autre erreur..  Ces erreurs sont certainement dues à un changement des répertoires de nos addon avec ce qui a du être enregistré quelque part … |
|  | Il y avait dans ce dossier 2 raccourcis vers 2 répertoires.. dont 1 n’existait plus…  On efface tout.. |
|  | On revient dans VSCode et on recharge notre premier projet addon qui entre temps a été changé de place.. et qui marchait plus.. pb plus haut..  On fait Ctrl Shift P et Blender : Start  Cette fois ci Blender se lance et l’addon est visible sur l’interface Blender… |
|  |  |
|  | On a cette sorte de raccourci qui vient de se rajouter dans le dossier user > AppData > addons.. |
|  | En clickant sur ce ‘raccouci’ on retrouve un répertoire avec les sources de l’addon..  En fait ce raccourci pointe sur le répertoire de travail initial de notre Addon.. ( Sans pour autant le montrer explicitement dans les propriétés du raccourci !! ) |
|  | Pour s’en convaincre , on rajoute un fichier dans le répertoire d’origine et on retrouve le fichier dans le répertoire pointé par le raccourci.. |
|  | Répertoire initial avec fichier dummy rajouté.. |
| On va charger un autre addon .. pour voir.. on ferme tous les folders dans visual studio.. |  |
|  | On recharge notre nouveau projet.. pardon.. Folder avec un autre addon en cours d’écriture et plein d’erreurs .. ( c’est la lib bpy dans le import bpy qui n’est pas trouvé dans l’editeur..)  **Pour retirer ces erreurs aller dans File>Preferences>Settting, chercher le parametre python.linting : mettre a false.**  On fait le Ctrl Shift P  Puis start Blender.. |
|  | Miracle, on voit dans Blender le nouvel Addon .. et le précédent… |
|  | Si on revient dans les Data cachées.. on trouve un autre raccourci qui pointe sur le nouveau AddonTestA |
|  | Image de tout ce qui se passe dans le répertoire d’origine.. |
|  | Répertoire de travail de l’addon pour VScode.. |
| **ON REMARQUE DONC QUE Tous les Addons travaillés via VSCode et l’extension Blender seront chargés dans Blender si on lance Blender via la commande de l’extension Blender dans VSCode ( Ctrl Shift P et Blender : Start ) : Important, il faut pas que les addon ait le même nom ( Même nom de répertoire …)** |  |

|  |  |
| --- | --- |
|  | **Il y a des erreurs dans le code pour manque de résolution de lib importée.**  **Pour retirer ces erreurs aller dans File>Préférences>Settting, chercher le paramètre python.linting : mettre a false** |
|  | Voilà.. |

**ANNEXE :**

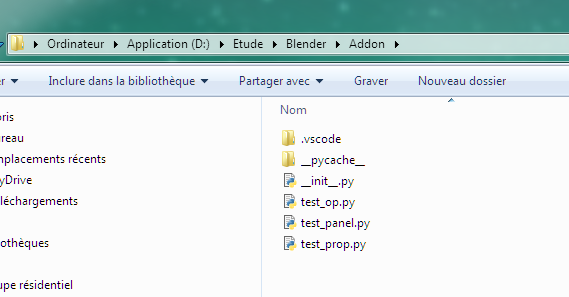
**Python..**

|  |  |
| --- | --- |
| **Indexation du code :**  L’indexation des lignes est très importante dans Python.. et on peut créer des tab avec le Tab ou avec des espaces.. Python ne supporte pas qu’on mélange.. et préfère que ce soit des espaces.. L’editeur VSCode permet de faire ça automatiquement via la touche Tab.. qui ne met pas un tab mais 4 espaces.. Par contre d’autres éditeurs.. peuvent ne pas faire ça et si le code se retrouve avec des espaces en guise de tab mélangé avec du vrai tab.. ça fait erreur dans Python… |  |
|  |  |
|  |  |

**Premier TestAddon En Vrac sans trop savoir le pourquoi / Comment...**

Exple projet dans l’environnement Visual Studio Code

Le répertoire se présentait ainsi .. puis on a changé l’organisation des répertoires et .. ça n’a plus marché..



|  |  |
| --- | --- |
|  | # This program is free software; you can redistribute it and/or modify  # it under the terms of the GNU General Public License as published by  # the Free Software Foundation; either version 3 of the License, or  # (at your option) any later version.  #  # This program is distributed in the hope that it will be useful, but  # WITHOUT ANY WARRANTY; without even the implied warranty of  # MERCHANTIBILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  # General Public License for more details.  #  # You should have received a copy of the GNU General Public License  # along with this program. If not, see <http://www.gnu.org/licenses/>.  bl\_info = {      "name" : "Test",      "author" : "Fred",      "description" : "Simple test addon",      "blender" : (2, 80, 0),      "version" : (0, 0, 1),      "location" : "View3D",      "warning" : "",      "category" : "Generic"  }  import bpy  from . test\_op import Test\_OT\_Operator  from . test\_op import Test\_OT\_Operator01  from . test\_op import Test\_OT\_Operator\_TestMove  from . test\_op import WM\_OT\_HelloWorld  from . test\_panel import Test\_PT\_Panel  from . test\_panel import OBJECT\_MT\_CustomMenu  from . test\_prop import MyProperties    classes = (      Test\_OT\_Operator,      Test\_PT\_Panel,      Test\_OT\_Operator01,      Test\_OT\_Operator\_TestMove,      MyProperties,      OBJECT\_MT\_CustomMenu,      WM\_OT\_HelloWorld      )  # remplacé par register / unregister explicite :  #register, unregister = bpy.utils.register\_classes\_factory(classes)  # voir  https://blender.stackexchange.com/questions/57306/how-to-create-a-custom-ui  #register / unregister explicite pour faire le travail a la main et rajouter aussi ce qu'on veut en plus  # comme ici l'instance de my\_tool..  def register():      from bpy.utils import register\_class      for cls in classes:          register\_class(cls)      bpy.types.Scene.my\_tool = bpy.props.PointerProperty(type=MyProperties)  def unregister():      from bpy.utils import unregister\_class      for cls in reversed(classes):          unregister\_class(cls)      del bpy.types.Scene.my\_tool |
|  | ##############################################################################  # ICI la partie interface IHM est codée  # Inspiré de :  # https://blender.stackexchange.com/questions/57306/how-to-create-a-custom-ui  ##############################################################################  import bpy  # ------------------------------------------------------------------------  #    Menus  # ------------------------------------------------------------------------  class OBJECT\_MT\_CustomMenu(bpy.types.Menu):      bl\_label = "Select"      bl\_idname = "OBJECT\_MT\_custom\_menu"      def draw(self, context):          layout = self.layout          # Built-in operators          layout.operator("object.select\_all", text="Select/Deselect All").action = 'TOGGLE'  # appel direct d'opérateur existant dans Blender          layout.operator("object.select\_all", text="Inverse").action = 'INVERT'  # en passant l'action a effectuer          layout.operator("object.select\_random", text="Random")  # ------------------------------------------------------------------------  class Test\_PT\_Panel(bpy.types.Panel):      bl\_idname = "Test\_PT\_Panel"      bl\_label = "Test Panel"      bl\_category = "Test Addon"      bl\_space\_type = "VIEW\_3D"      bl\_region\_type = "UI"      #bl\_context = "objectmode"                  #Si l'on veut le panel que dans context object..      @classmethod      def poll(self,context):          return context.object is not None      def draw(self, context):          layout = self.layout          scene = context.scene          mytool = scene.my\_tool   # l'objet my\_tool a été instancié dans le register..          row = layout.row()   # aligne les boutons sur la row          row.operator('view3d.cursor\_center', text="Center 3D cursor")          row.operator('view3d.test01', text="Test01")          layout.operator('object.move\_x', text="Test Move x") # ici un bouton a la ligne suivante          layout.separator()          layout.menu(OBJECT\_MT\_CustomMenu.bl\_idname, text="Presets", icon="SCENE")  # ici le menu défini plus haut          layout.separator()          layout.prop(mytool, "my\_bool")              # ici les propriétées définies dans test\_prop.py  qui vont trouver automatiquement leur représentation layout dans Blender          layout.prop(mytool, "my\_enum", text="")          layout.prop(mytool, "my\_int")          layout.prop(mytool, "my\_float")          layout.prop(mytool, "my\_float\_vector", text="")          layout.prop(mytool, "my\_string")          layout.prop(mytool, "my\_path")          layout.operator('wm.hello\_world') |
|  | import bpy  # voir  https://blender.stackexchange.com/questions/57306/how-to-create-a-custom-ui  # ------------------------------------------------------------------------  #    Scene Properties  # ------------------------------------------------------------------------  class MyProperties(bpy.types.PropertyGroup):      my\_bool: bpy.props.BoolProperty(          name="Enable or Disable",          description="A bool property",          default = False          )      my\_int: bpy.props.IntProperty(          name = "Int Value",          description="A integer property",          default = 23,          min = 10,          max = 100          )      my\_float: bpy.props.FloatProperty(          name = "Float Value",          description = "A float property",          default = 23.7,          min = 0.01,          max = 30.0          )      my\_float\_vector: bpy.props.FloatVectorProperty(          name = "Float Vector Value",          description="Something",          default=(0.0, 0.0, 0.0),          min= 0.0, # float          max = 0.1      )      my\_string: bpy.props.StringProperty(          name="User Input",          description=":",          default="",          maxlen=1024,          )      my\_path: bpy.props.StringProperty(          name = "Directory",          description="Choose a directory:",          default="",          maxlen=1024,          subtype='DIR\_PATH'          )      my\_enum: bpy.props.EnumProperty(          name="Dropdown:",          description="Apply Data to attribute.",          items=[ ('OP1', "Option 1", ""),                  ('OP2', "Option 2", ""),                  ('OP3', "Option 3", ""),                 ]          ) |
|  | import bpy  import bmesh  class Test\_OT\_Operator(bpy.types.Operator):      bl\_idname = "view3d.cursor\_center"      bl\_label = "Simple operator"      bl\_description = "Center 3d cursor"      def execute(self, context):          bpy.ops.view3d.snap\_cursor\_to\_center()          return {'FINISHED'}    class Test\_OT\_Operator01(bpy.types.Operator):      bl\_idname = "view3d.test01"      bl\_label = "Simple operator"      bl\_description = "desc - test01"      def execute(self, context):          #bpy.ops.view3d.snap\_cursor\_to\_center()          active\_object = bpy.context.view\_layer.objects.active          selected\_object = bpy.context.view\_layer.objects.selected          listSelected = []          for obj in selected\_object:              listSelected.append(obj)          bpy.ops.object.select\_all( action = 'SELECT' )          bpy.ops.object.origin\_set( type = 'ORIGIN\_GEOMETRY' )          bpy.ops.object.select\_all( action = 'INVERT' )          for obj in listSelected:              if obj.type == 'MESH':                  for vertex in obj.data.vertices:                      print("name :",obj.name," location :", obj.location , "  vertex : ", vertex.co)          verts = [(+1.0, +2.0, -1.0),                  (+1.0, -1.0, -2.0),                  (-1.0, -1.0, -1.0),                  (-1.0, +2.0, -1.0),                  (+1.0, +1.0, +1.0),                  (+1.0, -1.0, +1.0),                  (-1.0, -1.0, +1.0),                  (-1.0, +1.0, +1.0),                  ]          edges=[]          faces = [                      (0, 1, 2, 3),                      (4, 7, 6, 5),                      (0, 4, 5, 1),                      (1, 5, 6, 2),                      (2, 6, 7, 3),                      (4, 0, 3, 7),                  ]          mesh = bpy.data.meshes.new("Box")          mesh.from\_pydata(verts, edges, faces)  #        bm = bmesh.new()  #https://svn.blender.org/svnroot/bf-blender/trunk/blender/release/scripts/templates\_py/operator\_mesh\_add.py   #       for v\_co in verts:   #           bm.verts.new(v\_co)          # for t in bm.verts:          #     u=t          # print(list(bm.verts))          # bm.verts.ensure\_lookup\_table()          # print(bm.verts[0])              # for f\_idx in faces:          #     bm.faces.new([bm.verts[i] for i in f\_idx])          # bm.to\_mesh(mesh)          # mesh.update()          #  # add the mesh as an object into the scene with this utility module          from bpy\_extras import object\_utils          object\_utils.object\_data\_add(context, mesh, operator=self)          #bpy.ops.mesh.primitive\_plane\_add(size=2.0, calc\_uvs=True, enter\_editmode=False, align='WORLD', location=(0.0, 0.0, 0.0), rotation=(0.0, 0.0, 0.0))          mode = active\_object.mode          return {'FINISHED'}  class Test\_OT\_Operator\_TestMove(bpy.types.Operator):      # """My Object Moving Script"""      # Use this as a tooltip for menu items and buttons.      bl\_idname = "object.move\_x"        # Unique identifier for buttons and menu items to reference.      bl\_label = "Move X by One"         # Display name in the interface.      #bl\_options = {'REGISTER', 'UNDO'}  # Enable undo for the operator.      x=0      def execute(self, context):          self.report({'INFO'},"param x %d" % (self.x))          toto = self.x          # The original script          scene = context.scene          for obj in scene.objects:              obj.location.x += 1.0          return {'FINISHED'}            # Lets Blender know the operator finished successfully.  class WM\_OT\_HelloWorld(bpy.types.Operator):      bl\_label = "Print Values Operator"      bl\_idname = "wm.hello\_world"      def execute(self, context):          scene = context.scene          mytool = scene.my\_tool          # print the values to the console          print("Hello World")          print("bool state:", mytool.my\_bool)          print("int value:", mytool.my\_int)          print("float value:", mytool.my\_float)          print("string value:", mytool.my\_string)          print("enum state:", mytool.my\_enum)          return {'FINISHED'} |
|  |  |
|  |  |