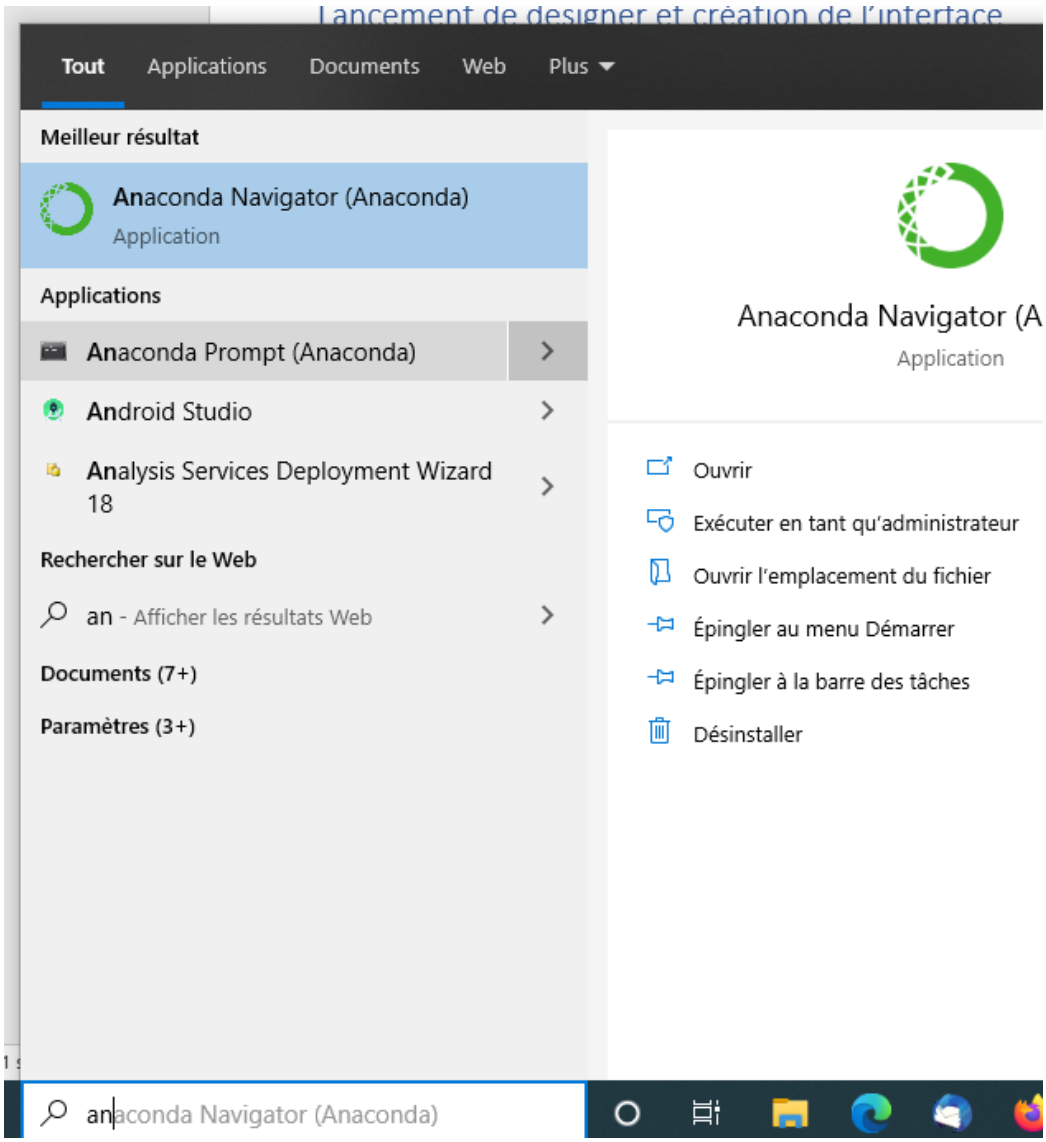
# PyQt5 tutoriel

## Installation de PyQt5

Depuis un terminal Anaconda prompt (en tant qu'administrateur) :



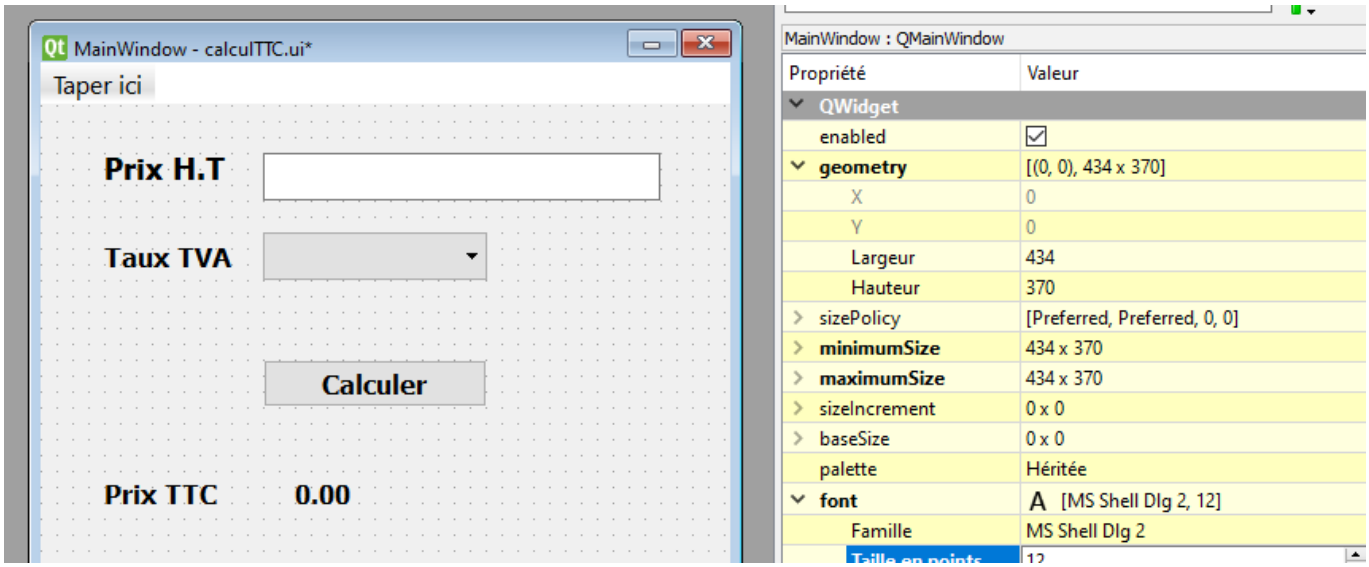

```
conda install -c anaconda pyqt
```

## Lancement de designer et création de l'interface

Depuis un terminal Anaconda prompt :

```
(base) C:\Users\user>designer
```

Créer par glisser déposer l'interface suivante :




| | **widget** | **nom** |
|---|---|---|
| Les zones de saisie : | Text Edit : | txtPrixHT et txtPrixTTC |
| Les zones de label : | Label : | labelHT, labelTaxe et labelTTC |
| Bouton de validation : | push button : | boutonOK |
| Liste déroulante : | combobox : | taxe |

Personnaliser ensuite la police (taille, famille) dans la fenêtre de propriété

# Création du script PyQt

La structure de base d'un fichier pyQt peut être associée au fichier ui du designer ou bien auto générée

### Par génération automatique :

Taper dans un terminal la commande suivante :

pyuic5 -x xxxxxxxx.ui -o xxxxxxxx.py

### Par association :

fichier pyqt_struct.py

```
import sys
from PyQt5 import QtGui, uic, QtCore, QtWidgets


class MyApp(QtWidgets.QMainWindow):
    def __init__(self):
        super(MyApp, self).__init__()
        QtWidgets.QMainWindow.__init__(self)
        uic.loadUi("xxxxxxxx.ui",self)


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MyApp()
    window.show()
    sys.exit(app.exec_())
```

# Peuplement de la liste déroulante :

L'objet combobox correspondant à la liste déroulante est nommé taxe.

Il sera initialisé dans le constructeur :

```
self.taxe.addItem("20")
self.taxe.addItem("5.5")
```

**Utilisation d'un conteneur en attribut :**

```
class MyApp(QtWidgets.QMainWindow,Ui_MainWindow):
    tab_taxes = [20, 5.5]
```

Initialisation dans le constructeur

```
for val in self.tab_taxes :
    self.taxe.addItem(str(val))
```

# Récupération de la valeur des zones de saisie :

```
Dans les zones de texte : txtPrixHT.toPlainText()

Dans la liste déroulante : self.taxe.currentText()
```

## Association du bouton de validation à une action :

Ajouter dans le constructeur l'événement associé au bouton qui lance la fonction calculPrixTTC :

```
self.boutonOK.clicked.connect(self.CalculPrixTTC)
```

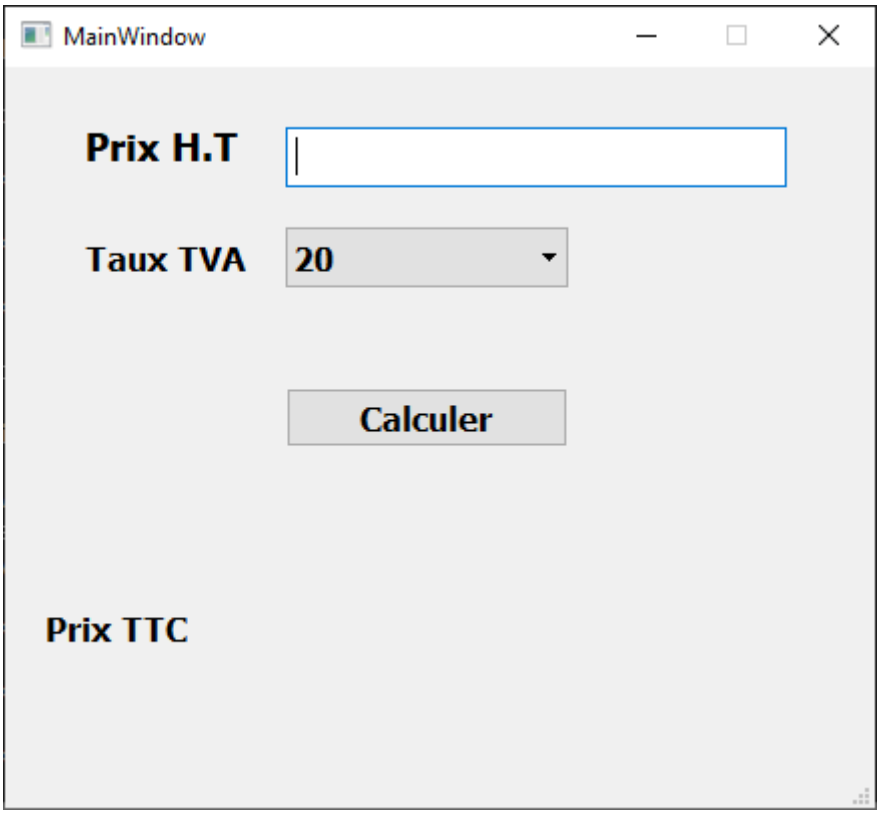
Fonction CalculPrixTTC :

```
def CalculPrixTTC(self):
    prix = int(self.txtPrixHT.toPlainText())
    taxe = float(self.taxe.currentText())

    prixTTC = prix + ((taxe / 100) * prix)
    prixTTC_string = "Le prix total TTC est : " + str(prixTTC)

    self.txtPrixTTC.setText(prixTTC_string)
```

## Lancement de l'application :

Dans le dossier contenant le script lancer la commande :

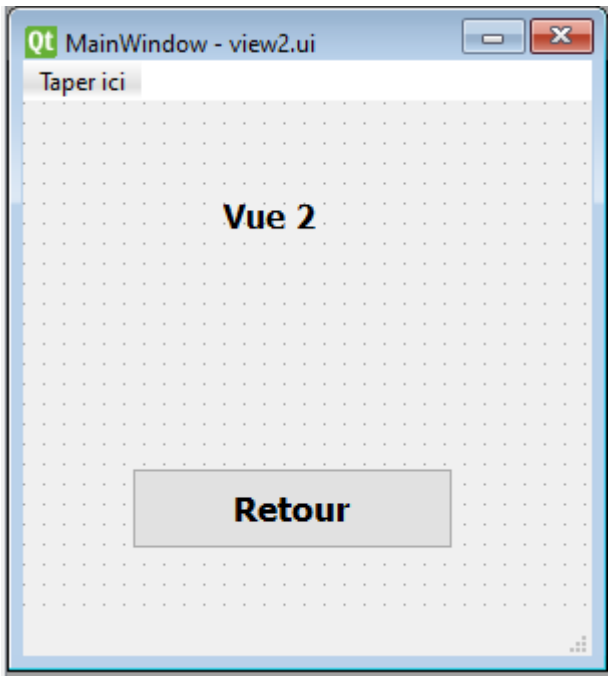
```
python qt_calculTTC.py
```



## Communication ente vues :

**View1**


**View2**




Programme principal

```
class View1(QtWidgets.QMainWindow):

    def __init__(self):
        super().__init__()
        uic.loadUi('view1.ui', self)
        self.btnValid.clicked.connect(self.openVue2)

    def openVue2(self):
        self.hide()
        Vue2 = View2(self)
        Vue2.show()

class View2(QtWidgets.QMainWindow):

    def __init__(self, parent=None):
        super().__init__(parent)
        uic.loadUi('view2.ui', self)
        self.btnRetour.clicked.connect(self.retourVue1)

    def retourVue1(self):
        self.parent().show()
        self.close()


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = View1()
    window.show()
    sys.exit(app.exec_())
```

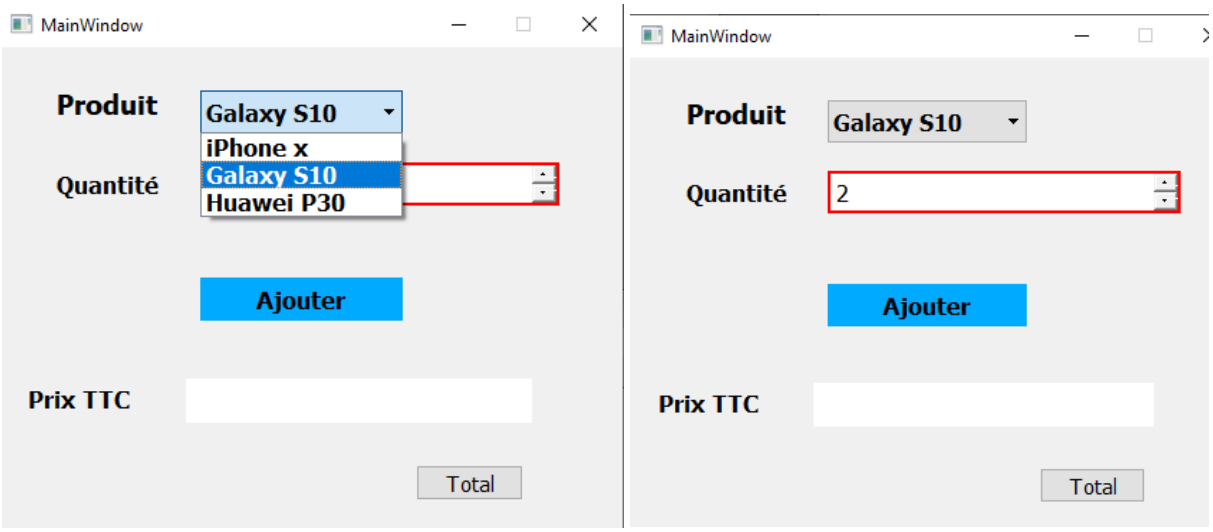
## Exercices

On souhaite améliorer l'application pour simuler un carnet de commandes :

Une liste de produits est donnée sous la forme d'un dictionnaire ;

```
liste_produits = {'iPhone x':870, 'Galaxy S10': 765, 'Huawei P30': 670}
```

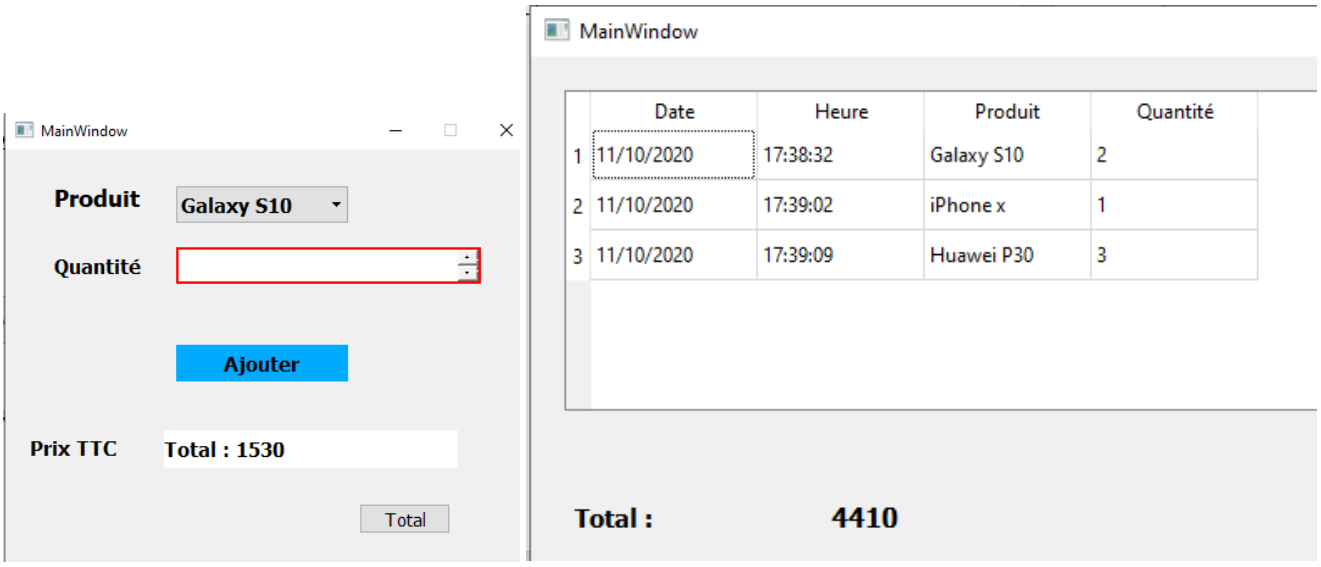
L'interface graphique proposera un choix de produits dans une liste déroulante et demandera à saisir une quantité de produits.




Un bouton Ajouter permettra de réinitialiser la quantité et de faire une nouvelle saisie.

Le calcul du total se fera automatiquement et le montant correspondant sera enregistré dans un tableau qui permettra d'alimenter un objet QTableWidget dans la vue suivante.

Un bouton Total affichera une nouvelle vue présentant l'ensemble de la commande dans un tableau en ajoutant la date et l'heure de chaque commande ainsi que le montant total.



## Annexes :

### Alimenter un QtableWidget par un array

```
def qtableFromArray(self, array, qtable):
    nbRow = len(array)
    nbCol = len(array[0])
    qtable.setRowCount(nbRow)
    qtable.setColumnCount(nbCol)
    for i in range(nbRow):
      for j in range(nbCol):
          qtable.setItem(i,j,QTableWidgetItem(str(array[i][j])))
```

### Affichage des dates et heures

```
from PyQt5.QtCore import QDate, Qt

now = QDate.currentDate()
print(now.toString('d.M.yy'))


from PyQt5.QtCore import QTime, Qt

time = QTime.currentTime()
print(time.toString('h.m.s'))
```