TP2 : Gestion des événements, utilisation de nouveaux objets (recherche dans l'aide en ligne), disposer des objets dans une fenêtre, contrôle de données

1 Composants de sélection

Ergonomie :Composant de sélections

Il permet de choisir un ensemble de données de façon exclusive ou non. On distingue les boutons de sélections(radio ou à cocher), les listes de sélections, les listes.

Utiliser les boutons de sélections pour des choix fréquents et peu nombreux.(toujours visibles, mais occupant beaucoup de place, donc utiles aux informations pertinentes)

Employer la liste de sélection lorsque la place est réduite et les choix peu fréquents.

Utiliser la liste lorsque l'ensemble des choix possibles est variable (mais difficile d'utilisation car on utilise des barres de défilement.)

par défaut, les « checkBox » ne sont pas exclusifs et les « radioButton » le sont. Les groupes de boutons permettent de rassembler ces 2 deux types d'objet.

```
class Widget_lecure_sexe(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)

        ???=QVBoxLayout() #declaration du conteneur
        self.GroupSexe = QGroupBox(self)
        self.GroupSexe.setTitle("Sexe")
        layout_s_sexe=???
        self.checkBoxMasculin = QRadioButton("Homme")
        self.checkBoxFeminin = QRadioButton("Femme")
        self.checkBoxFeminin.setChecked(True)

        layout_s_sexe.addSpacing ???
        layout_s_sexe.addWidget( self.checkBoxMasculin)
        layout_s_sexe.addWidget( self.checkBoxFeminin)
        layout_s_sexe.addStretch ???

        self.GroupSexe.setLayout(???)
        self.comboBoxSexe = QComboBox(self)
        self.comboBoxSexe.insertItem(0,"Homme")
        self.comboBoxSexe.insertItem(1,"Femme")

        layout_sexe.addWidget(???)
        layout_sexe.addWidget(???)
        self.setLayout(???)
```

Exercice 1 : compléter le programme ci dessus pour réaliser l'interface ci dessous
*** disposer correctement le lineEdit et les 2 label(s) dans un layout**
*** disposer tout ces éléments dans un layout**
*** insérer un espace fixe de 100 avant le label**
*** insérer un strech de coefficient 2 avant le slider (après lineEdit)**
*** insérer un strech de coefficient 1 avant le lcdNumber (après le slider)**

Rajouter la ligne ci dessous pour modifier le style de l'objet `GroupSexe`

```
 self.GroupSexe.setStyleSheet("   background-color: qlineargradient(x1: 0, y1:
0, x2: 0, y2: 1, stop: 0 #E0E0E0, stop: 1 #FFFFFF); border: 2px solid
gray;border-radius:5px; ");
```

2 Introduction à la partie événementielle

Une fois l'interface dessinée, les objets placés : il faut définir les événements et les actions.
Sous Qt, il y a 2 étapes :
définir les slots (ce sont des fonctions (méthodes des objets))
définir les connexions, on définit quel objet utilise quel slot, et sur quel événement.

Définition des slots ; Rajouter le code suivant dans le programme python, juste avant d'afficher la fenêtre.
définir les connexions ; Rajouter le code suivant dans le programme python

```
Mon_bouton.connect(Mon_bouton, SIGNAL("clicked()"),fonction1)
```

ce sont des connecteurs sur des objets : l'objet qui déclenche l'appel est le premier paramètre (un objet peut être composé de plusieurs objets différents) , le signal qui déclenche l'événement (ex: cliqué)est le second paramètre, le slot appelé (exemple : fonction1) est le troisième paramètre.
Le seul élément compliqué à comprendre est l'utilisation du mot clef : **self** devant un nom d'objet, devant une méthode. Il permet de rendre **public** l'objet et l'utiliser partout dans les méthode.

```
def fonction1():
      Mon_bouton.setText("appel de la fonction1") #met ce texte dans le bouton
```

Voici la nouvelle fonction : elle permet d'afficher les zones de travail possible à un sportif en fonction de sa fréquence cardiaque de travail et maximum.

```
    def fonction1(self):
        fcMax,ok1 = self.textEdit1.text().toDouble() #frequence cardiaque Max
        fcWork=self.MonSpinBox.value()  #frequence cardiaque de la vie
quotidienne
        if ( ok1 ) :
            fcReserve=fcMax-fcWork
            f1=fcWork+0.5*fcReserve
            f2=fcWork+0.6*fcReserve
            f3=fcWork+0.7*fcReserve
            f4=fcWork+0.8*fcReserve
            f5=fcWork+0.9*fcReserve
            texteAffiche="<br /> endurance Fondamentale entre "+str(f1)+"et
"+str(f2)+" <br />"
            texteAffiche+="<br />endurance Active entre "+str(f2)+"et
"+str(f3)+" <br />"
            texteAffiche+="<font color=\"#ff0000\"> travail 20 mn => 1H30 et
plus</font> "
            texteAffiche+="<br /><br /> resistance douce entre "+str(f3)+"et
"+str(f4)+" <br />"
            texteAffiche+="<font color=\"#ff0000\">   travail 20 => 40
mn</font> "
            texteAffiche+="<br /> <br /> resistance dure entre "+str(f4)+"et
"+str(f5)+" <br />"
            texteAffiche+="<font color=\"#ff0000\">  travail 6 => 8 mn / recup
3 => 4 mn * (2/4) fois </font>"
            self.MonLabel3.setText(texteAffiche)
        else :
            self.MonLabel3.setText(" erreur sur la saisie de la frequence max
")
```

Rajouter des connecteurs pour faire l'appel du slot (méthode,fonction) sur 2 autres événements : changement d'une des valeurs des fréquences.

```
self.textEdit1.connect(self.textEdit1, SIGNAL("textChanged (const
QString&)"),self.fonction1)
self.MonSpinBox.connect(self.MonSpinBox,SIGNAL("valueChanged(int)"),self.foncti
on1)
```

C'est dans cette partie que l'on voit l'intérêt de la partie événementielle : avec peu de code, on peut rendre une interface conviviale, on distingue bien 2 élèments : la vue (creation de l'interface) et le traitement des événements (contrôleur) qui modifie l'interface en fonction de ces événements. La gestion des données est faite par le modèle (partie non traitée) .

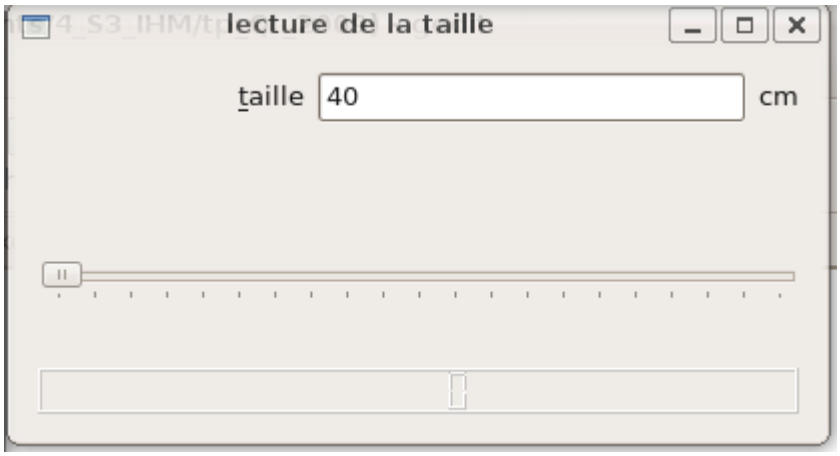
définition de chacun de ces 3 mots :
* **le modèle** : c'est là que l'on stocke les données ou, plus généralement, là ou se passent les évènements (tels qu'on le verra ci-dessous)
* **la vue** : c'est la partie visuelle, basée sur les données du modèle et qui va agir différemment en fonction des évènements envoyés par celui-ci. Il peut y avoir plusieurs vues ayant pour un seul modèle (c'est bien le but !) et chacune d'entre elles adopteront

des comportements spécifiques lorsqu'elles recevront les évènements générés par les modèles.
* **le contrôleur** : les vues interagissant avec le modèle, elles ont parfois (souvent) besoin de modifier les données. Le contrôleur sert à faire la liaison entre le modèle et la vue. C'est notamment lui qui, dans l'exemple ci-dessous, s'occupe du traitement des données avant de les transmettre au modèle. Le traitement des données étant souvent spécifique à chaque vue, il y en a généralement un par vue.

3 exemple de programmation événementielle

Reprendre les exercices du TP1 :
exo1_4.py

exo1_5.py



Exercice 2 :
A partir du code ci dessous, rajouter correctement les évènements pour modifier le textEdit en même temps que le slider, le dial et le LCDnumber .

Rechercher les événements dans l'aide sur Qt (doc.trolltech.com)

```
    def modif_dial_poids(self):
        self.LCD_poids.display(self.dial_poids.value())
        texte=QString()
        texte.setNum(self.dial_poids.value())
         self.lineEdit_poids.setText(texte)

    def modif_lineEdit_poids(self):
```

```
         val,ok=self.lineEdit_poids.text().toInt()
         if(val>self.dial_poids.maximum()) :
                 self.dial_poids.setMaximum(val)
            self.dial_poids.setValue(val)
        else :
             self.dial_poids.setValue(self.lineEdit_poids.text().toInt()[0])

    def modif_slider_taille(self):
        self.LCD_taille.display(self.slider_taille.value())
        texte=QString()
        texte.setNum(self.slider_taille.value())
         self.lineEdit_taille.setText(texte)

    def modif_lineEdit_taille(self):
         val=self.lineEdit_taille.text().toInt()[0]
         if(val>self.slider_taille.maximum()) :
                 self.slider_taille.setMaximum(val)
            self.slider_taille.setValue(val)
        else :
             self.slider_taille.setValue(self.lineEdit_taille.text().toInt()
[0])

    def modif_checkBoxSexe(self):
        if self.checkBoxMasculin.isChecked():
            self.comboBoxSexe.setCurrentIndex(0)
        if self.checkBoxFeminin.isChecked():
            self.comboBoxSexe.setCurrentIndex(1)

    def modif_comboBoxSexe(self):
        if self.comboBoxSexe.currentIndex()==0:
            self.checkBoxMasculin.setChecked(True)
        if self.comboBoxSexe.currentIndex()==1:
            self.checkBoxFeminin.setChecked(True)
```

Sujet :

Le but de ce tp est de faire un petit logiciel pour calculer le poids idéal en fonction de son sexe, sa taille et lui indiquer sa catégorie en fonction de son poids :

Index de corpulence = Indice de Masse Corporelle (IMC) = Body Mass Index (BMI)

BMI (Body Mass Index) = Poids(kg) / Taille(m)^2

en fonction de cet indice afficher le message suivant :

Situation pondérale	Femme	Homme
Maigreur (insuffisance pondérale)	<19.1	< 20.7
Poids idéal	19.1 - 25.8	20.7 - 26.4
à la limite du surpoids	25.8 - 27.3	26.4 - 27.8
Surpoids	27.3-32.3	27.8 - 31.1
Obésité	> 32.3	> 31.1

Poids idéal (IBW) Formule de Lorentz (1929)

- Femme = Taille(cm) - 100 - [Taille(cm) - 150] / 2
- Homme = Taille(cm) - 100 - [Taille(cm) - 150] / 4

poids idéal exprimé en kg
Conditions de l'utilisation de cette formule :

- âge supérieur à 18 ans ;
- taille entre 140 et 220 cm (55 à 87 inch)

Si le poids est supérieur au poids théorique on affiche « surcharge pondérale » sinon « poids correct ».

Exercice 3 : Modifier votre code de façon à obtenir l'interface suivante :

Il est demandé de soigner l'ergonomie, de commenter son code, et d'être capable de l'expliquer, voir de le modifier rapidement (lors de l'évaluation, il vous sera demandé de déplacer un objet).

Essayer d'éviter un comportements non souhaité des objets (avec par exemple la méthode minimumSize)

aide :
Placer les groupes d'objets précédents dans un layout de type « QGridLayout »

```
layout_principal=QGridLayout() #declaration du conteneur principale
layout_principal.addLayout(layout_poids,0,0)
#........ on ajoute ici les autres layouts ; le 0,0 correspond à la position
dans la grille
self.setLayout(layout_principal)
```

Le layout est comme une grille que l'on remplie, (remarque : c'est bien expliqué sur http://www.siteduzero.com/tutoriel-3-11302-0-positionner-ses-widgets-avec-les-layouts.html)

```
layout_principal.addLayout(layout_sexe,0,2)
```

- Réaliser l'affichage du résultat des formules précédentes (énoncé) :

Insérer le code suivant dans le slot (fonction) appelé par les objets qui modifie la taille, le poids, l'âge et le sexe.

```
      def affiche_informations(self):
            texte=QString()
            texte2=QString()
            div_sexe, poids, taille,poids_th, bmi,age,kcal=0,0,0,0,0,0,0
            if self.checkBoxMasculin.isChecked():
                  div_sexe=4.0
            else:
                  div_sexe=2.0

            poids=float(self.lineEdit_poids.text())
            taille=float(self.lineEdit_taille.text())
            age=float(self.lineEdit_age.text())
            poids_th=float(taille-100-(taille-150)/div_sexe)
            texte2=str(poids_th)
            texte+="<b><u>Formule de Lorentz : </u></b>"
            texte+="poids ideal :"
            texte+=texte2
            if(poids_th*1.1 < poids):
                  texte+="<font color=\"#ff0000\">   surcharge ponderale</font>"
            texte+="<br><b><u>Index de corpulence =IMC=BMI :</u></b>"
            bmi=poids/(taille*taille)*10000
            texte2=str(bmi)
            texte+=texte2
            if(((bmi <= 19.1) and div_sexe ==2) or (bmi <=20.7 and div_sexe
==4)) :
                  texte2="<font color=\"#ff0000\">   Maigreur (insuffisance
ponderale)</font>"
            if((bmi <= 25.8 and bmi > 19.1  and div_sexe ==2) or (bmi <= 26.4
and bmi > 20.7  and div_sexe ==4)):
                  texte2="<font color=\"#ff0000\">   Poids ideal</font>"
            if((bmi <= 27.3 and bmi > 25.8 and div_sexe ==2) or(bmi <= 27.8 and
bmi > 26.4 and div_sexe ==4)):
                  texte2="<font color=\"#ff0000\">   a la limite du
surpoids</font>"
            if(bmi <= 32.3 and bmi > 27.3 and div_sexe ==2) or(bmi <=  31.1 and
bmi > 27.8 and div_sexe ==4):
                  texte2="<font color=\"#ff0000\">   Surpoids</font>"
            if(((bmi > 32.3) and div_sexe ==2) or (bmi > 31.1 and div_sexe ==4))
:
                  texte2="<font color=\"#ff0000\">  Obesite</font>"
            texte+=texte2
            self.textLabel_information.setText(texte)
```