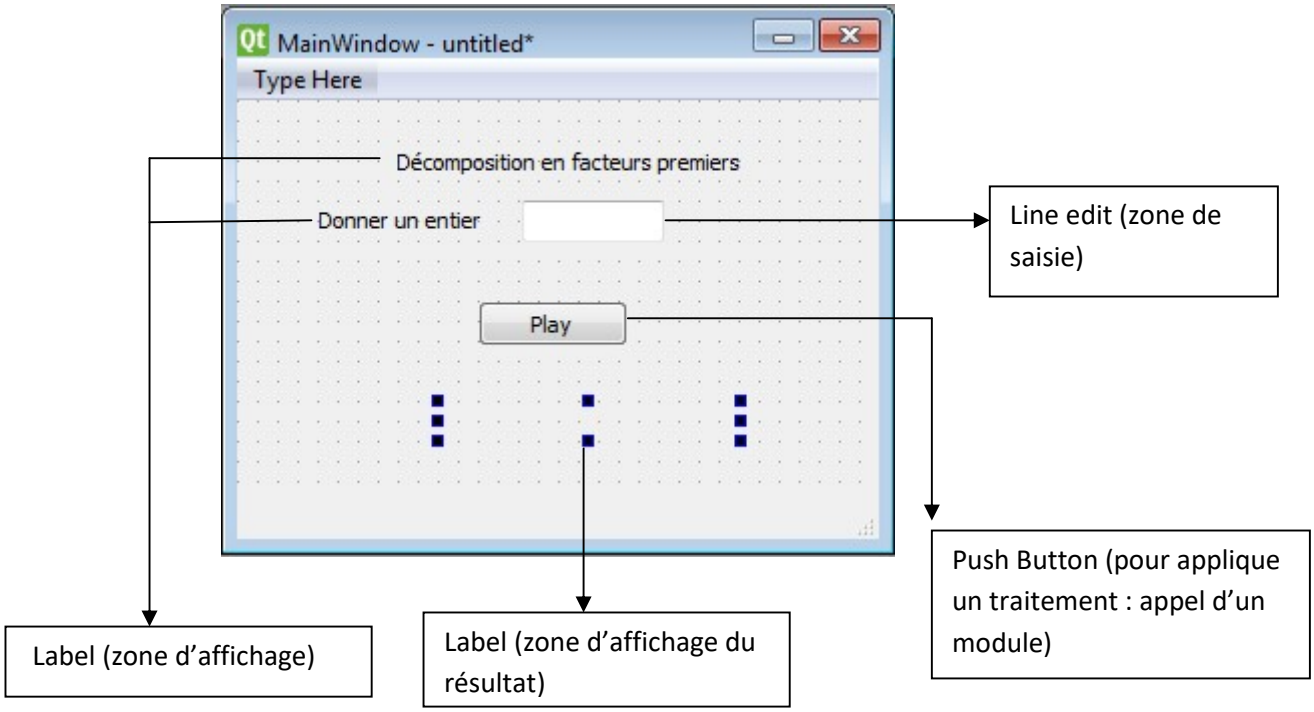
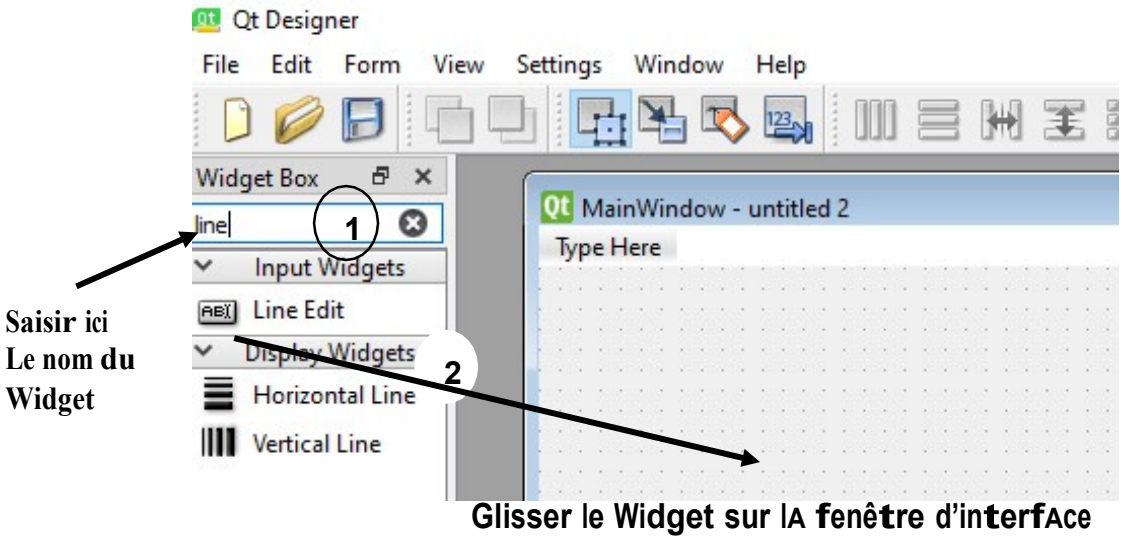
Application1 :

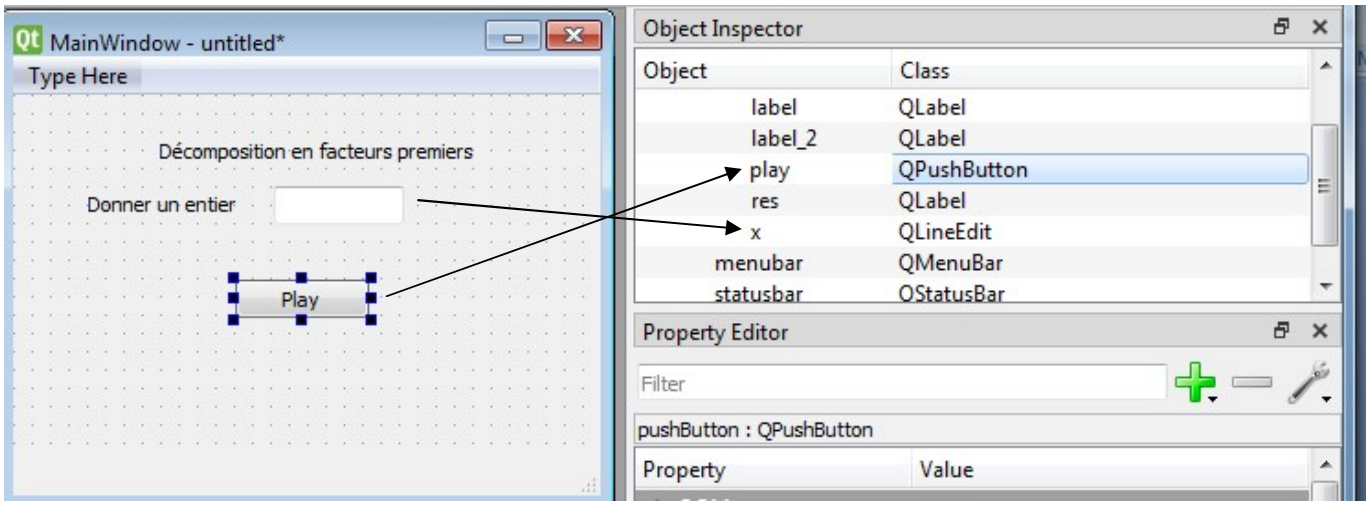
Décomposition d'un entier > 1 en facteurs premiers.



Pour insérer les Widgets :

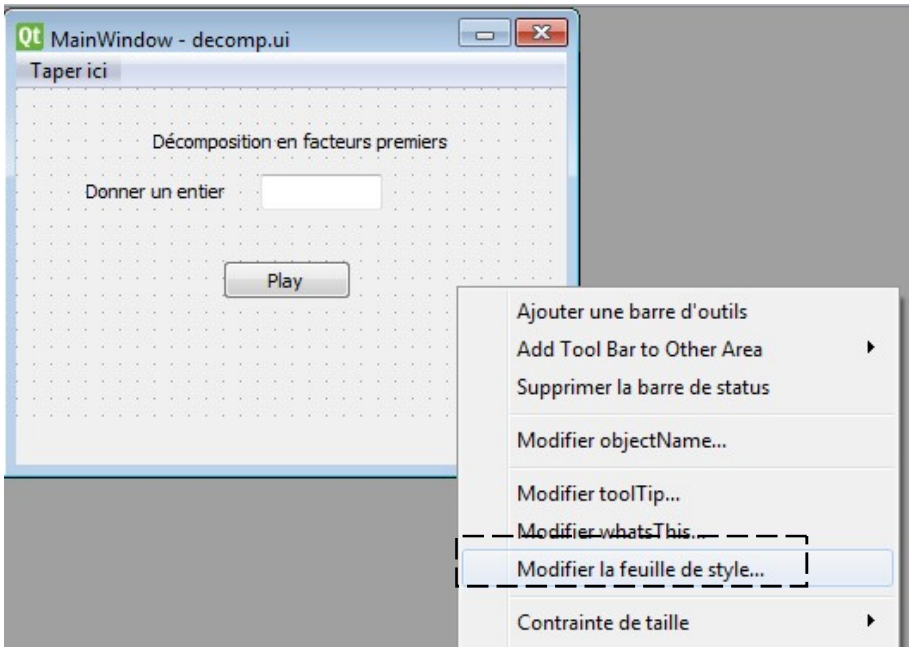


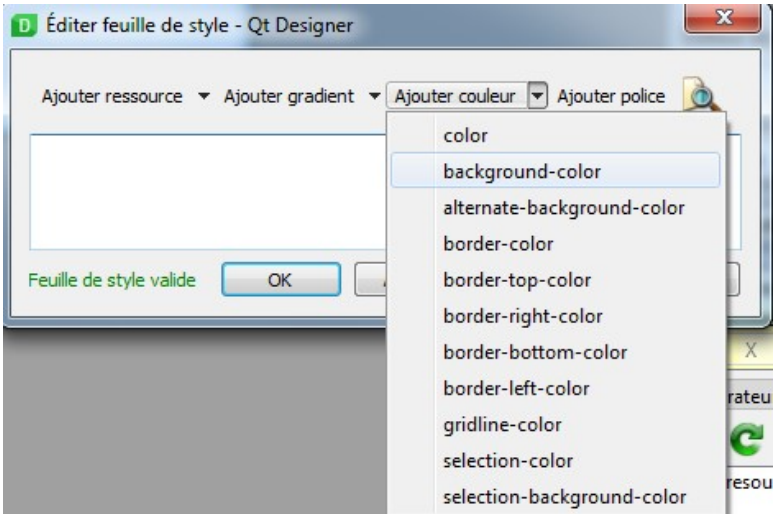
Nommer les widgets (exemple : zone de saisie → x ; zone d'affichage → res ; bouton → play)



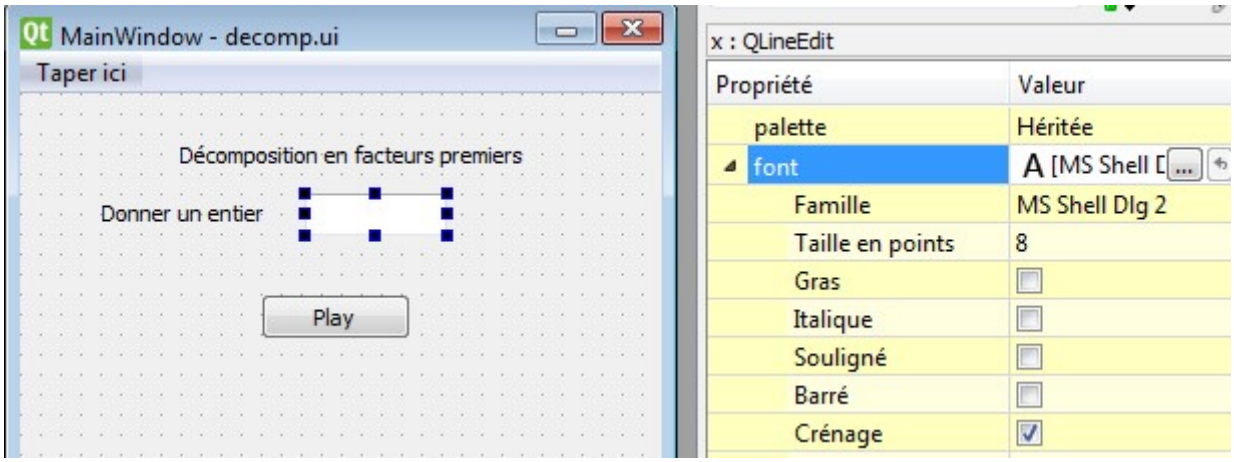
- Pour appliquer des mises en forme sur la fenêtre (exp : changer les couleurs)

Clic droit dans la fenêtre (ou dans le widget à modifier)



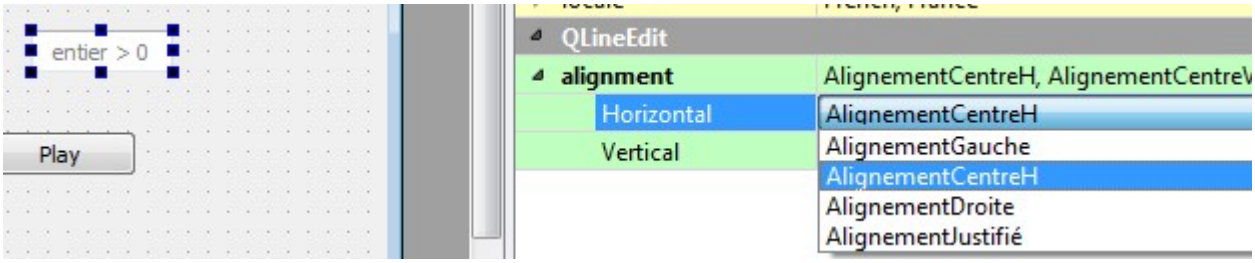


- On peut modifier aussi les caractéristiques d'un widget.



- Insérer un texte explicatif dans une zone de saisie :

- Changer l'alignement d'une zone de saisie (Line edit)



- Changer le nom de la fenêtre :

Les méthodes à utiliser pour écrire un programme simple :

Champ de texte - LineEdit

Un **champ de texte** (aussi appelé *champ de saisie de texte*) est une zone de texte dans laquelle l'utilisateur peut entrer ou modifier un texte.

`text()` **:** Renvoie du texte contenu dans le champ de texte.
Renvoie : texte contenu dans le champ de texte
Type renvoyé : string (str)

Etiquette - Label

Une **étiquette** (en anglais *label*) est un composant de type `QLabel` qui permet d'afficher un texte par l'utilisateur, mais que le programme peut faire évoluer (changer).

`setText(text)` **:** Modifie le texte de l'étiquette (label).

```
from PyQt5 import QtWidgets, uic
app = QtWidgets.QApplication([])
dlg = uic.loadUi("decomp.ui")
dlg.x.setFocus()
def decomp_facteurs(x):
    ch = ""
    i = 2
    while(x > 1):
        if x % i == 0:
            ch = ch + str(i) + "*"
            x = x // i
        else:
            i = i + 1
    return ch[:len(ch) - 1]
def facteurs(ch):
    x = int(ch)
    if x <= 1:
        dlg.res.setText("ERREUR DE SAISIE")
        dlg.x.setText("")
        dlg.x.setFocus()
    else:
        ch = decomp_facteurs(x)
        dlg.res.setText(str(x) + " = " + ch)
dlg.play.clicked.connect(lambda: facteurs(dlg.x.text()))
dlg.show()
app.exec()
```

Placer le curseur initialement dans la zone de saisie.

Tester la validité de la valeur saisie

Si la valeur saisie est correcte, on appelle la fonction à exécuter et afficher le résultat dans la zone convenable

Appel de la fonction avec paramètre

2éme forme : appel sans paramètre

```
dlg.play.clicked.connect(facteurs)
```

NB : Bien sûr, on doit prendre en compte les modifications au niveau de l'entête de la fonction.

Forme générale

from PyQt5 import QtWidgets, uic #appel de la bibliothèque
app=QtWidgets.QApplication([]) #app est le nom d'une variable(on peut le changer)
dlg= uic.loadUi(" ") #dlg est le nom logique du fichier physique "exemple1.ui"
……
……..
……..
……..

Les modules : Traitements à réaliser suite à l'action clicked.connect

#Programme Principal
dlg ……..clicked.connect(.....)
dlg.show()
app.exec()

Appel de la fonction à exécuter :
1) (decomp) : appel d'une fonction non paramétré.
2) (lambda :decomp(dlg.x.text)) : appel d'une fonction avec paramètres

Le nom du boutton (exemple : play)

Application2 :

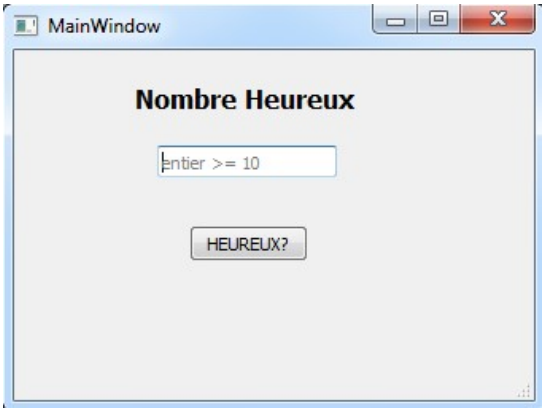
Un nombre heureux est un entier strictement positif, qui, lorsqu'on additionne les carrés de chacun de ses chiffres, puis on additionne les carrés des chiffres de la somme obtenue et ainsi de suite, on obtient un entier à un seul chiffre et est égal à 1.

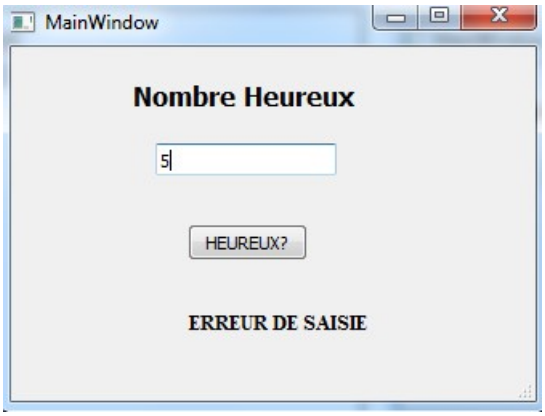
Exemples :

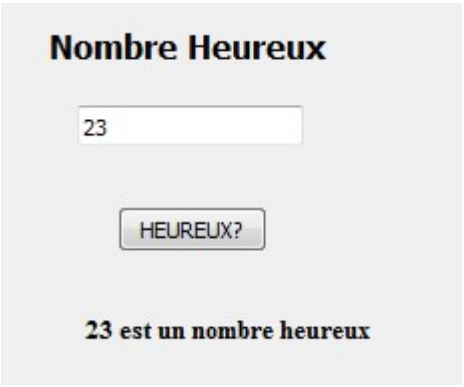
- 10 est un nombre heureux car $1^2 + 0^2 = 1$
- 23 est un nombre heureux car $2^2 + 3^2 = 13$ / $1^2 + 3^2 = 10$ / $1^2 + 0^2 = 1$
- 109 est un nombre heureux car $1^2+0^2+9^2=82/8^2+2^2=68/6^2+8^2=100/1^2+0^2+0^2=1$

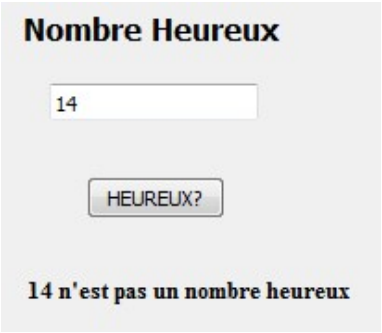
Saisir un entier x >= 10, et vérifier s'il est heureux ou non.

```
from PyQt5 import QtWidgets, uic
app = QtWidgets.QApplication([])
dlg = uic.loadUi("heureux.ui")
dlg.x.setFocus()

def heureux(n):
    while n >= 10:
        s = 0
        while n != 0:
            r = n % 10
            s = s + r * r
            n = n // 10
        n = s
    return n == 1

def verif(ch):
    if int(ch) < 10:
        dlg.res.setText(" ERREUR DE SAISIE")
        dlg.x.setText("")
        dlg.x.setFocus()
    else:
        v = heureux(int(ch))
        if v:
            dlg.res.setText(ch + " est un nombre heureux")
        else:
            dlg.res.setText(ch + " n'est pas un nombre heureux")

dlg.verif_heureux.clicked.connect(lambda: verif(dlg.x.text()))
dlg.show()
app.exec()
```









Application 3:

Dans le cadre d'une campagne publicitaire, une société commerciale a décidé d'organiser chaque semaine un jeu de chance pour ses clients.

Le principe du jeu consiste à calculer le nombre de chance à partir du numéro de téléphone du client donné et d'afficher le message "***Félicitations, vous avez gagné.***" dans le cas où ce nombre est **premier** ou le message "***Désolé, vous n'avez pas gagné.***" dans le cas contraire.

Sachant que :

- Le numéro de téléphone devrait commencer par 2, 4, 5 ou 9.
- Le nombre de chance est la somme de chaque chiffre du numéro de téléphone multiplié par son indice avec l'indice du premier chiffre est 0.
- Un nombre premier est un nombre qui est divisible par 1 et par lui-même.

Exemple :

Donner le numéro : ***29234560***

Le programme affiche : ***Désolé, vous n'avez pas gagné.***

En effet, le nombre de chance est égal à 99 qui n'est pas un nombre premier.

99 = 2 * 0 + 9 * 1 + 2 * 2 + 3 * 3 + 4 * 4 + 5 * 5 + 6 * 6 + 0 * 7 c'est la somme de chaque chiffre du numéro de téléphone multiplié par son indice :

Numéro téléphone	2	9	2	3	4	5	6	0
Indice	0	1	2	3	4	5	6	7

Ci-après, un algorithme de la fonction **"Chance"** à exploiter pour résoudre le problème posé.

```
Fonction Chance (Ch : Chaine) : Chaine
DEBUT
    Si NON ( Estnum ( Ch ) ET long ( Ch ) = 8 ET Ch[0] ∈ ["2","4","5","9"] ) Alors
        msg ← "Vérifier le numéro de téléphone !"
    Sinon
        msg ← "Désolé, vous n'avez pas gagné."
        s ← 0
        Pour i de 0 à long ( Ch ) - 1 Faire
            s ← s + valeur ( Ch [i] ) * i
        Fin Pour
        Si premier (s) Alors
            msg ← "Félicitation, vous avez gagné."
        FinSi
    FinSi
    Retourner msg
FIN
```

```
from PyQt5 import QtWidgets,uic
app = QtWidgets.QApplication([]) # app est le nom d'une variable
dlg = uic.loadUi("prototype bac sc.ui") # liaison entre nom logique (dlg) et nom physique
dlg.x.setFocus() #placer le curseur dans la zone texte x
def chance(ch):
    if not(ch.isdigit() and len(ch)==8 and ch[0] in ["2","4","5","9"]):
        msg="Vérifier le numéro de téléphone!"
    else:
        msg="Désolé, vous n'avez pas gagné"
        s=0
        for i in range(len(ch)):
            s=s+int(ch[i])*i
        if premier(s):
            msg="Félicitation, vous avez gagné."
    return msg
def premier(x):
    i = 2
    while(i<= x // 2) and (x % i !=0):
        i = i + 1
    return i > x // 2
def play():
    ch = dlg.x.text()
    dlg.res.setText(chance(ch))
dlg.b.clicked.connect(lambda:play()) # appel d'une fonction sans paramètres
dlg.x.returnPressed.connect(lambda:play()) #utilisation de la touche entrée du clavier pour exécuter la fonction
dlg.show()
app.exec()
```

Exemples d'exécution :

Application 4 :

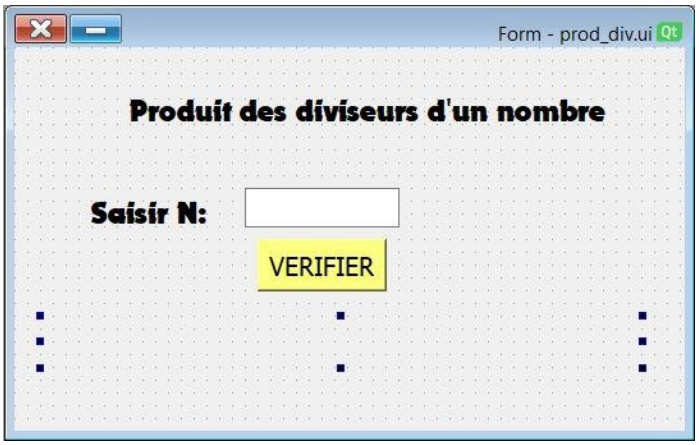
Etant donné un entier **N** qui vérifie la propriété suivante : «**Le produit des diviseurs de Nsauf lui-même est égal à N**».

Exemples :

- **N = 10** vérifie cette propriété car le produit de ses diviseurs est égal à lui-même.En effet, **1 * 2 * 5 = 10**.
- **N = 12** ne vérifie pas cette propriété car le produit de ses diviseurs n'est pas égal à lui-même. En effet, **1 * 2 * 3 * 4 * 6 = 144**.

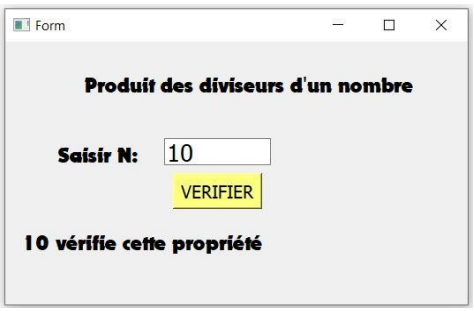
Pour vérifier qu'un nombre N vérifie cette propriété ou non, on doit créer tout d'abord une interface graphique suivante, comportant les éléments suivants :

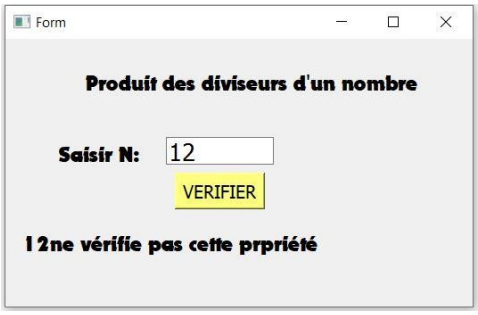
- **un label lbl** pour le **titre**.
- **un label** nommé **nb**.
- **un label** nommé **res**, réservé pour l'affichage du résultat.
- **Une zone de saisie** nommé "**n"** permettant la saisie de **N**.
- **Un bouton** contenant le texte " **vérifier** " (à nommer **btn)**.

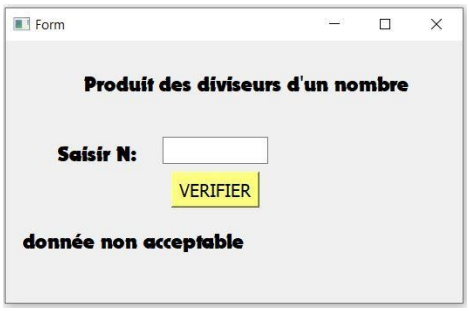


Travail demandé :

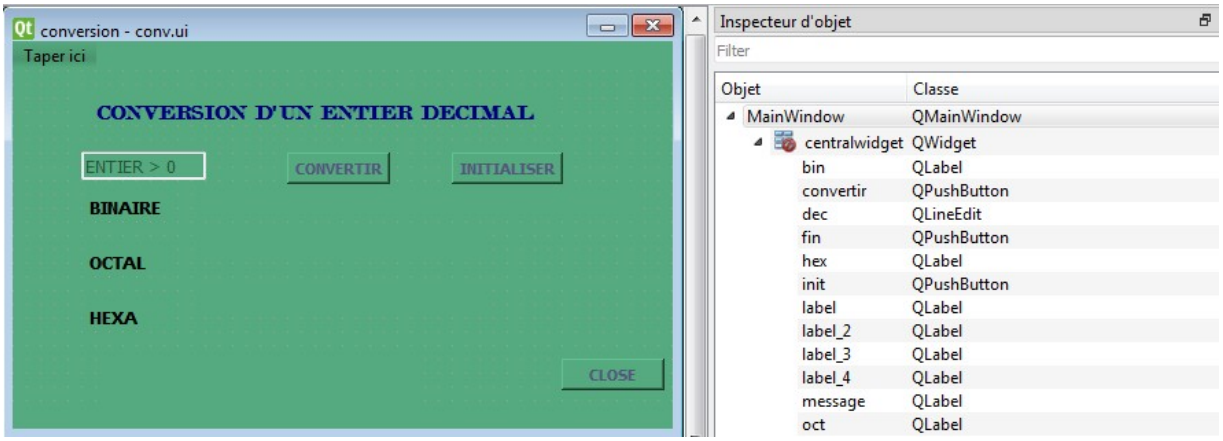
1) Concevoir une interface graphique comme illustré ci-dessus et l'enregistrer, dans votre dossier de travail, sous le nom "**propriété.ui**".
2) Développer en Python la fonction "**valide** ", (qui vérifie la propriété ci-dessus), dans un **programme et l'enregistrer sous le nom "pro_diviseur.py", dans votre dossier de travail.**
3) Développer la fonction "**affiche**", qui s'exécute à la suite d'un clic sur le bouton "**VERIFIER**", permettant de :
   - ✓ Vérifier si la donnée **N** est composée uniquement par des chiffres et **> 0** puis en apportant les appels et les modifications convenables à afficher :
     - Le message **« N vérifie cette propriété »** si la fonction "valide" retourne vrai -via un **label de l'interface déjà créée-.**
     - Sinon afficher « **N ne vérifie cette propriété** » et vider la zone de texte de **N**.
   - ✓ Si la donnée contient des erreurs pendant la saisie, vider la zone de texte de **N** et afficher « **Donnée non acceptable !!!!** » comme résultat.







Application 5:




```
from PyQt5 import QtWidgets, uic
app = QtWidgets.QApplication([])
dlg = uic.loadUi("conv.ui")
dlg.dec.setFocus()
def conv10_b(n, b):
    ch = ""
    while n > 0:
        r = n % b
        if r < 10:
            ch = str(r) + ch
        else:
            ch = chr(r + 55) + ch
        n = n // b
    return ch
def traitement_convrsion(ch):
    if int(ch) < 0:
        dlg.message.setText("ERREUR DE SAISIE")
        dlg.dec.setText("")
        dlg.dec.setFocus()
    else:
        if dlg.message.text != "":
            dlg.message.setText("")
        dlg.bin.setText(conv10_b(int(ch), 2))
        dlg.oct.setText(conv10_b(int(ch), 8))
        dlg.hex.setText(conv10_b(int(ch), 16))
def traitement_initialisation():
    dlg.dec.clear()
    dlg.dec.setFocus()
    dlg.bin.clear()  # setText("")
    dlg.oct.clear()  # setText("")
    dlg.hex.clear()  # setText("")
dlg.convertir.clicked.connect(lambda:
traitement_convrsion(dlg.dec.text()))
dlg.dec.returnPressed.connect(lambda:
traitement_convrsion(dlg.dec.text()))
dlg.init.clicked.connect(lambda:
traitement_initialisation())
dlg.fin.clicked.connect(dlg.close)
dlg.show()
app.exec
```

Application 6:

Le code binaire reflété (RBC), également appelé simplement binaire réfléchi (RB) ou Le code Gray, est un ordre du système numérique binaire tel que deux valeurs successives ne diffèrent que dans un seul bit (chiffre binaire). Pour passer d'un nombre (en code Gray) au suivant, on ne change à la fois qu'un seul bit en procédant de la manière suivante :

- Si le nombre de bits à 1 est pair : on change le dernier bit.
- Si le nombre de bits à 1 est impair : on change le bit à gauche, du bit 1 le plus à droite.

Regarder les exemples suivants pour bien comprendre comment le code Gray fonctionne.

Exemple1 :

Soit le nombre = 110101, le nombre de bits à 1 est égale à 4, donc pair, on change le dernier bit et le nombre suivant est : 110100

Exemple2 :

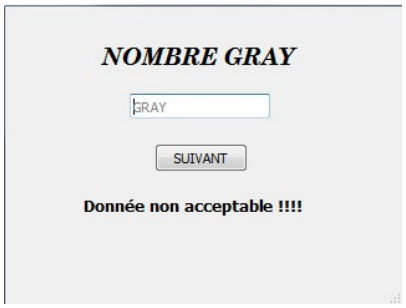
Soit le nombre = 110100, le nombre de bits à 1 est égale à 3, donc impair, on cherche le bit 1 le plus à droite puis on change le bit qui se trouve à sa gauche, le nombre suivant est 111100.

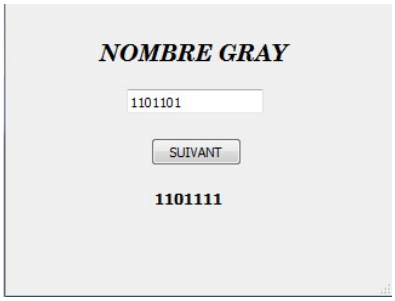
Travail demandé :

1) Concevoir une interface graphique comme illustré ci-dessous et l'enregistrer, dans votre dossier de travail, sous le nom "**gray.ui**".

Objet	Classe
MainWindow	QMainWindow
centralwidget	QWidget
gray	QLineEdit
label	QLabel
res	QLabel
suiv	QPushButton
menubar	QMenuBar
statusbar	QStatusBar

2) Développer la fonction "**affiche**", qui s'exécute à la suite d'un clic sur le bouton "**SUIVANT**", permettant de :
   a. Vérifier si la valeur saisie est composée uniquement par des chiffres binaires puis en apportant les appels et les modifications convenables à afficher :
   - Nombre gray suivant selon le principe décrit en haut, dans une zone d'affichage (label de l'interface déjà créée).
   - Si la donnée contient des erreurs pendant la saisie, vider la zone de texte et afficher « **Donnée non acceptable !!!!** » comme résultat.






```
from PyQt5 import QtWidgets,uic
app = QtWidgets.QApplication([])
dlg = uic.loadUi("gray.ui")
dlg.gray.setFocus()
def freq(c,ch):
    f = 0
    for i in range(len(ch)):
        if ch[i] == c:
            f = f + 1
    return f
def binaire_suivant(ch):
    f = freq("1",ch)
    if f % 2 == 0:
        if ch[len(ch)-1] == "0":
            ch = ch[:len(ch)-1] + "1"
        else:
            ch = ch[:len(ch)-1] + "0"
    else:
        p = position_droite("1",ch)
        if ch[p-1] == "0":
            ch = ch[:p-1] + "1" + ch[p:]
        else:
            ch = ch[:p-1] + "0" + ch[p:]
    return ch
def position_droite(c,ch):
    p = len(ch)-1
    while(ch[p] != c):
        p = p - 1
    return p
def verif(ch):
    i = 0
    while (i<len(ch)) and (ch[i] in ("0","1")):
        i = i + 1
    return i == len(ch)
def gray_suivant(ch):
    if not verif(ch):
        dlg.gray.clear()
        dlg.res.setText("Donnée non acceptable
!!!!")
        dlg.gray.setFocus()
    else:
        ch = binaire_suivant(ch)
        dlg.res.setText(ch)
dlg.suiv.clicked.connect(lambda:gray_suivant(d
lg.gray.text()))
dlg.show()
app.exec()
```