utbm
université de technologie
Belfort-Montbéliard

UV : MI44

Compte rendu MI44 : Cryptographie

Groupe du projet :
LOUKILI Younes
BLLADI Ismail

Table des matières

I. Introduction

Afin de mieux assimiler les notions de cryptographie (FEISTEL et RSA), vue en cours, il nous a été demandé pendant notre séance de travaux pratiques, d'élaborer des programmes dans le langage de notre choix. Ces derniers consistent d'une part de faire le chiffrement et déchiffrement « FEISTEL » et d'autre part celui de « RSA »

L'objectif du travail demandé est de simuler une communication sécurisée par RSA via l'échange d'un mot de passe. Ce dernier sera utilisé pour décrypter le message envoyé.

Notre travail sera composé de 5 fichiers d'extension « .py », nommés :

- Crypto.py : où l'on trouve toutes les fonctions utilisées pour le chiffrement et déchiffrement, ainsi que des fonctions intermédiaires qui nous ont facilités la tâche
- C.py : Programme qui génère les clés de l'expéditeur A et du destinataire B dans un fichier texte (donneesA.txt et donneesB.txt)
- A.py : Programme qui récupéré les données de l'expéditeur A à partir du fichier « donneeA.txt »
- B.py : Programme qui récupéré les données du destinataire B à partir du fichier « donneeB.txt »
- Main.py : c'est le fichier client, on l'en va afficher la communication entre A et B

II. Environnement technique

a. Langage utilisé :

Le langage qu'on a choisi pour programmer dans notre travail est appelé « Python ». C'est un langage de programmation objet, multiparadigme et multiplateformes. Il favorise la impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

b. Environnement de développement :

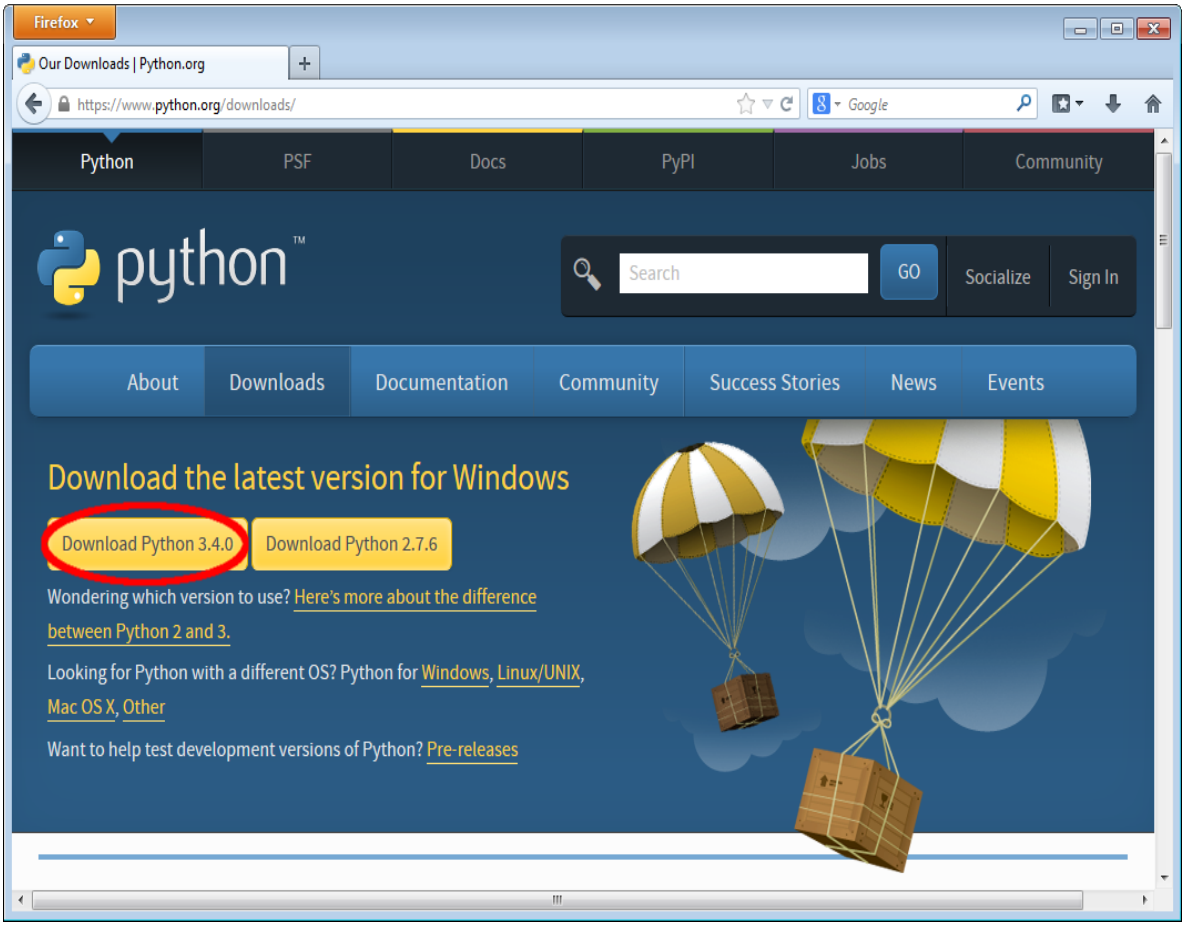
PyCharm est un environnement de développement intégré (IDE) utilisé pour programmer en Python.

Il offre l'analyse de code, un débogueur graphique, la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, et supporte le développement web avec Django.

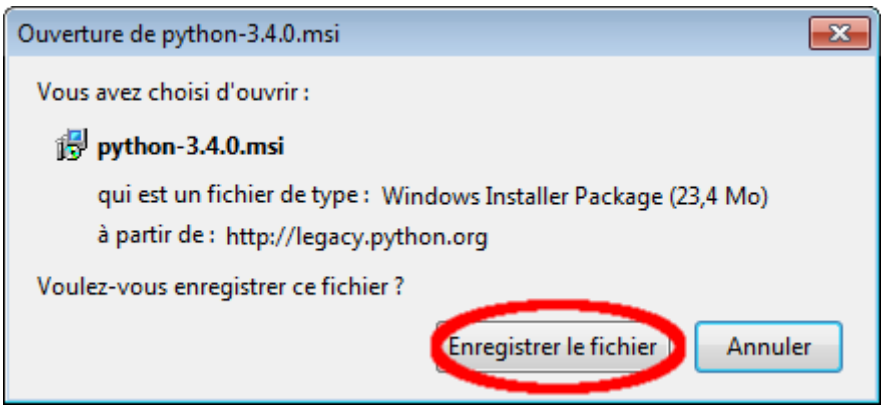
Il est développé par l'entreprise tchèque JetBrains. Il est multi-plateforme et fonctionne sous Windows, Mac OS X et Linux. Il est décliné en édition professionnelle, réalisé sous licence propriétaire, et en édition communautaire réalisé sous licence Apache.

c. Comment installer Pycharm ?

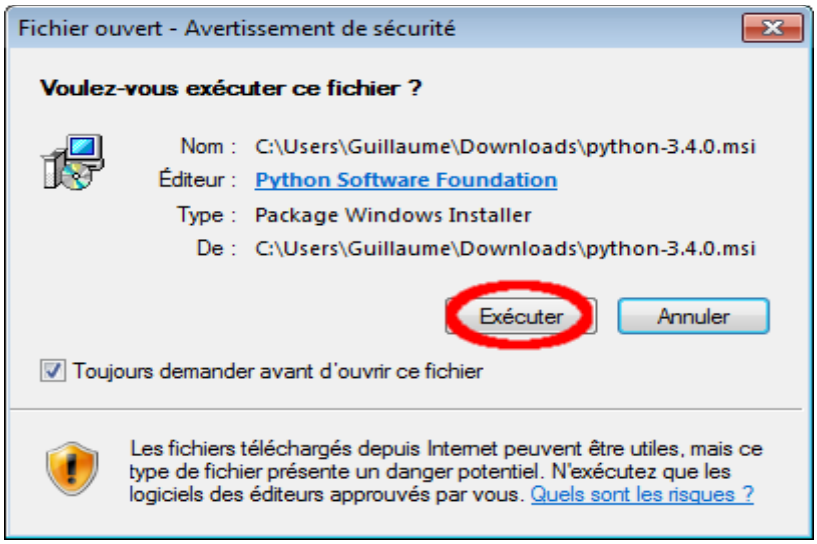
Nous allons tout d'abord installer Python 3. Pour cela, il faut télécharger le programme d'installation disponible sur le site internet de Python à l'adresse suivante : http://www.python.org/downloads/ , puis cliquer sur le "Download Python 3.x.y".



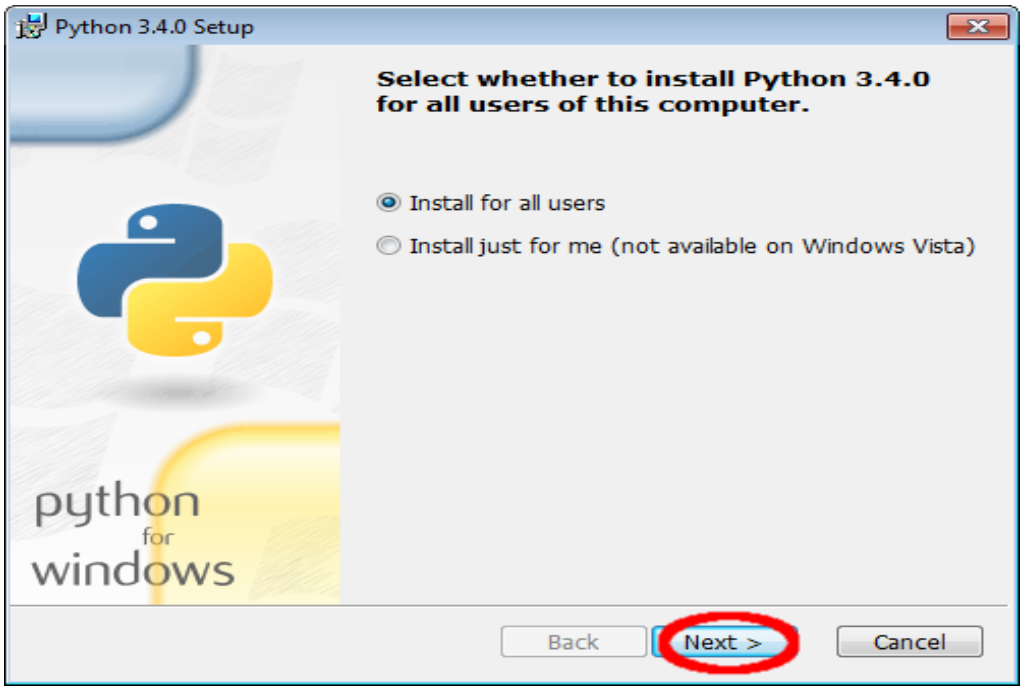
Enregistrez le programme d'installation sur votre disque dur.



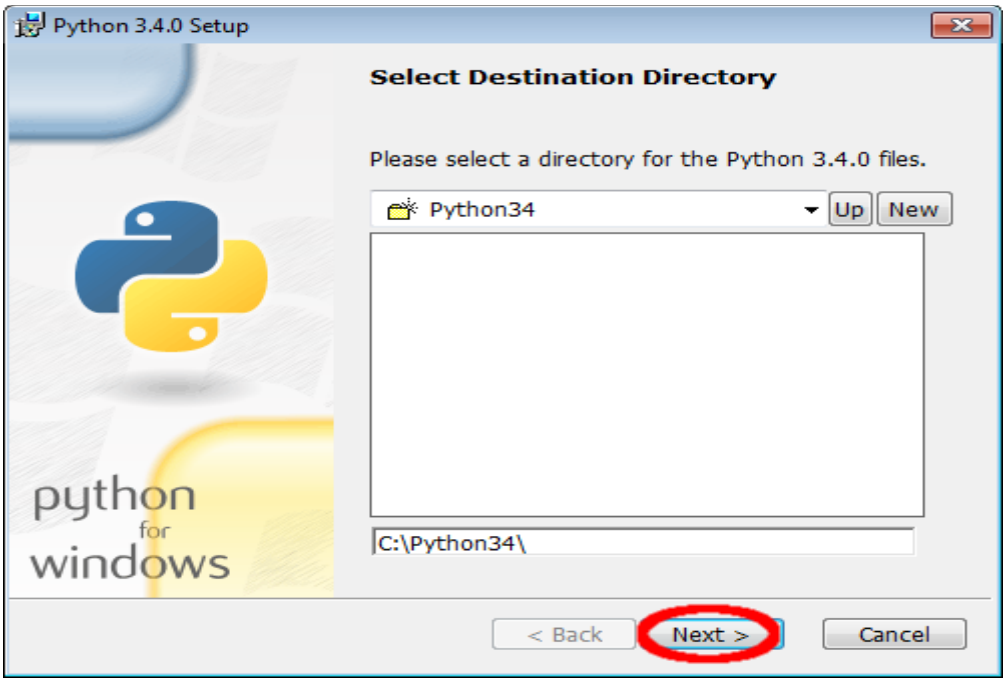
Lancez le programme d'installation qui vient d'être téléchargé.



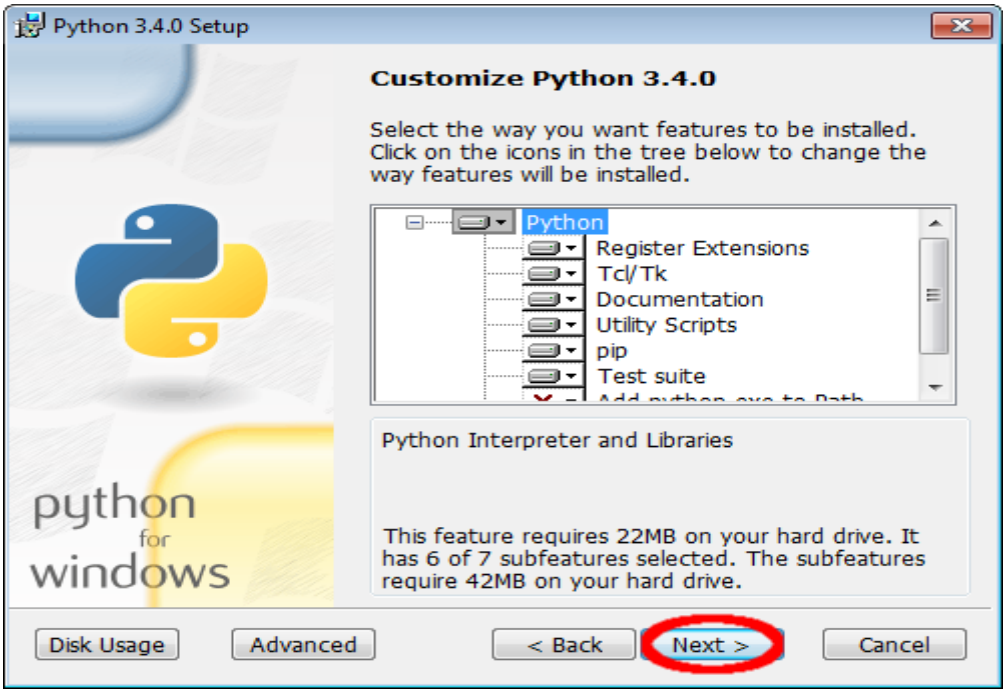
Choisissez “Exécuter” lors de l’avertissement de sécurité.



Cliquez sur “Next >”.



Cliquez sur “Next >”.

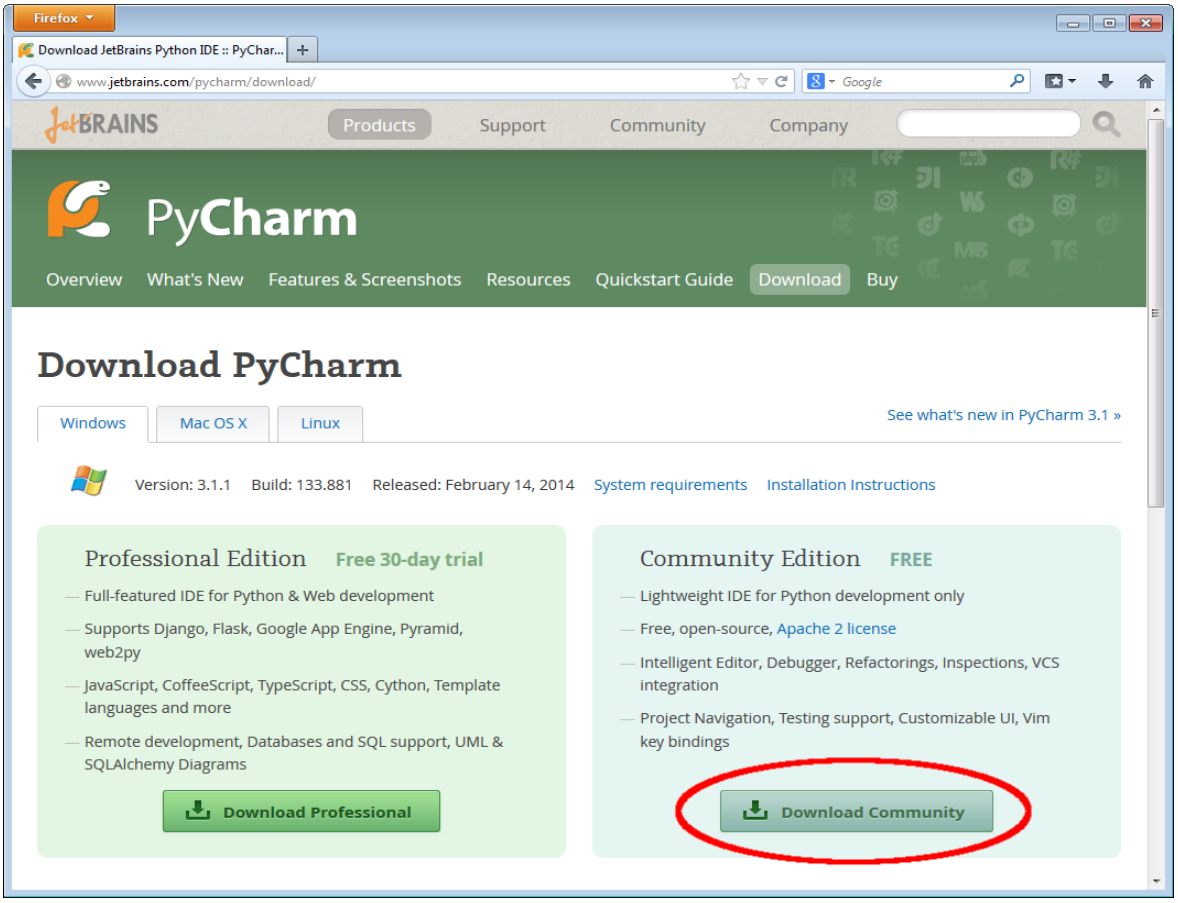


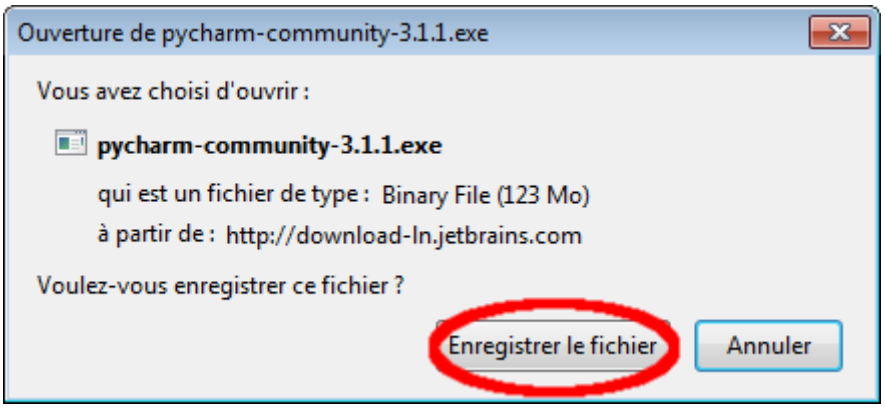
Cliquez sur “Next >”.

Python 3 est à présent installé.

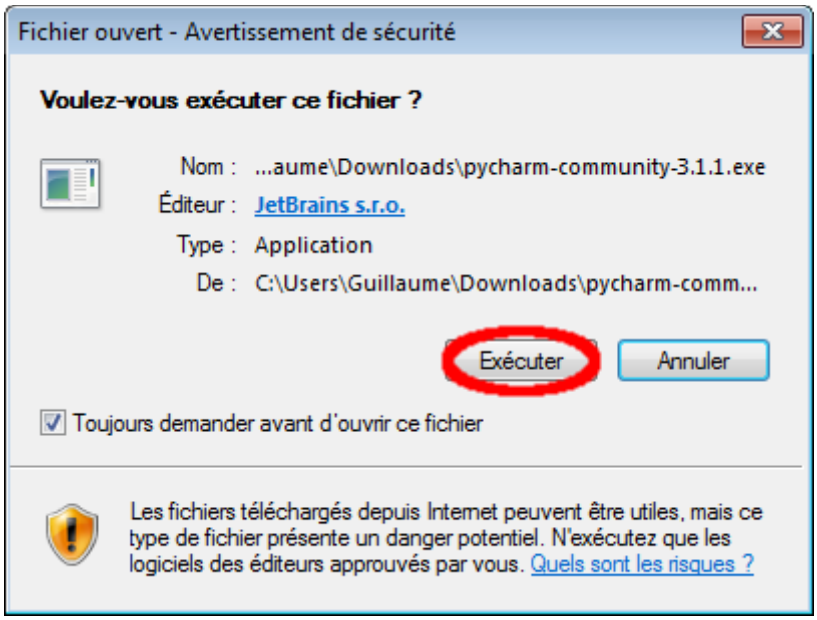
Après, rendez-vous sur la page de téléchargement du logiciel
: http://www.jetbrains.com/pycharm/download/ .

Cliquez sur “Download Community” afin de télécharger la version Community de PyCharm.



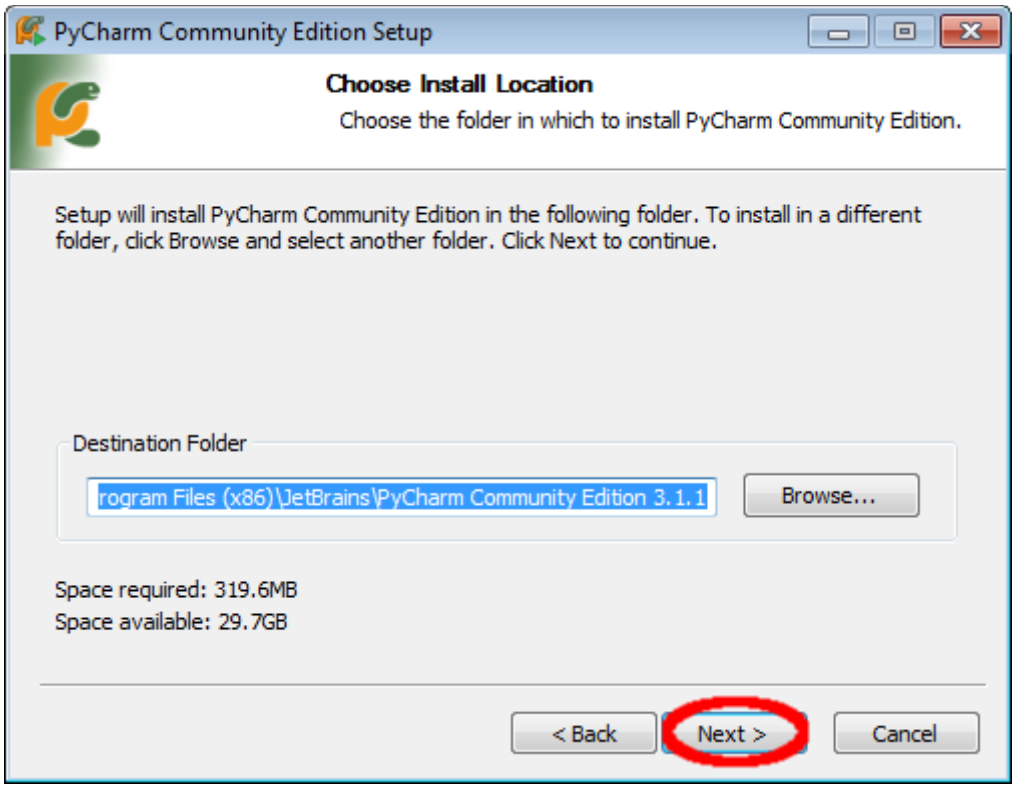


Enregistrez le programme d’installation.

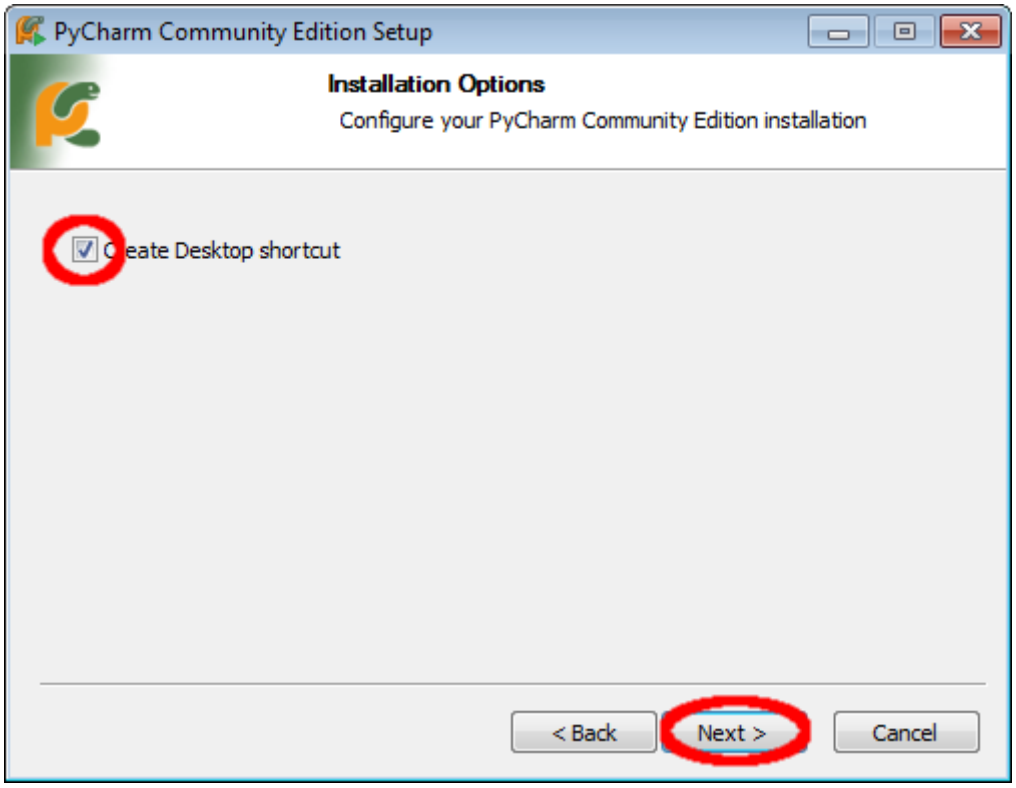


Lancez le programme d'installation qui vient d'être téléchargé, et cliquer sur "Exécuter" lors de l'avertissement de sécurité.

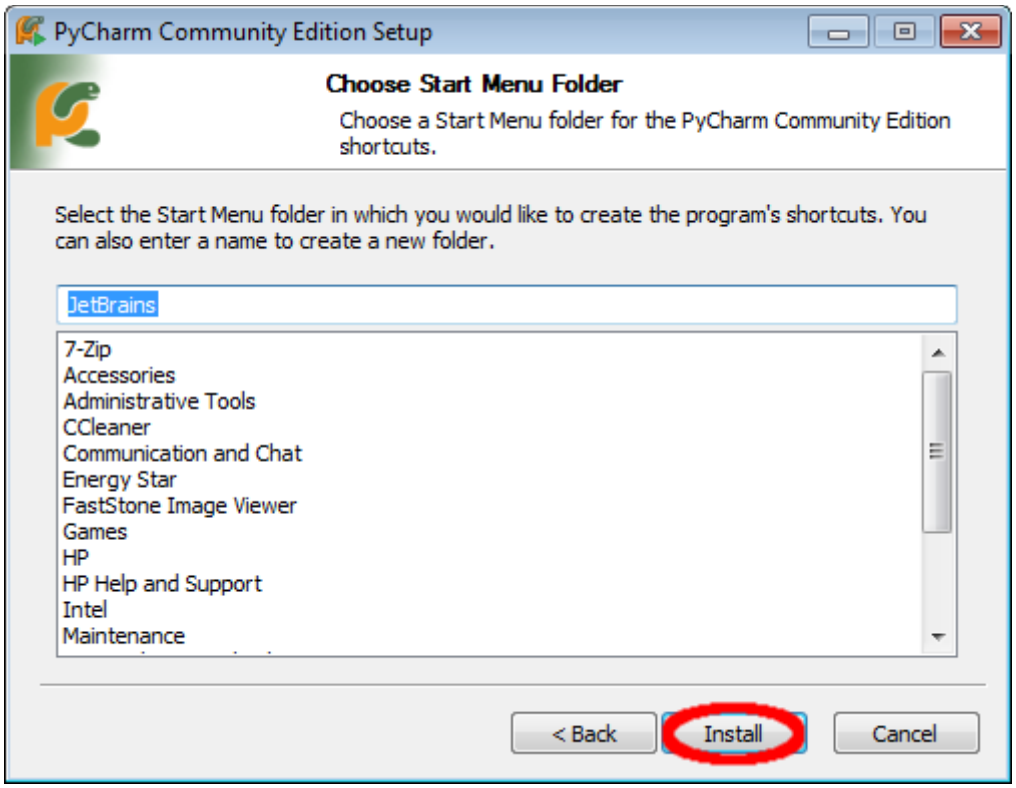
Cliquez sur "Next >"



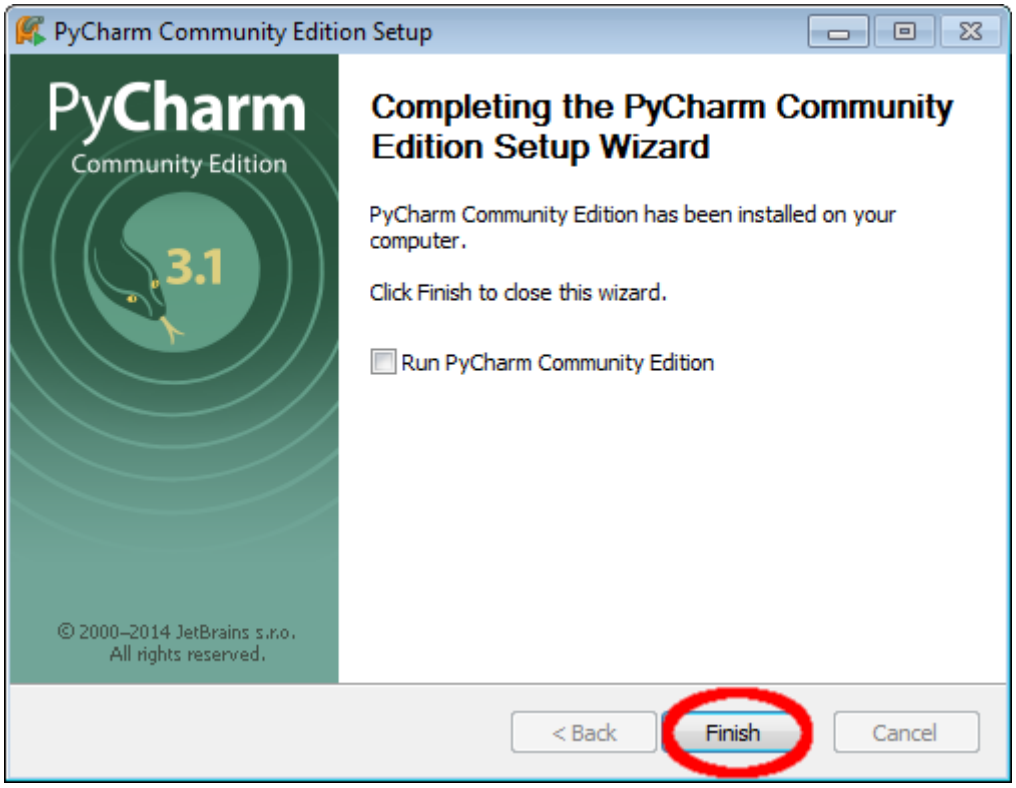
Cliquez sur “Next >”



Cochez la coche pour mettre un raccourci sur le bureau, et cliquez sur “Next >”.



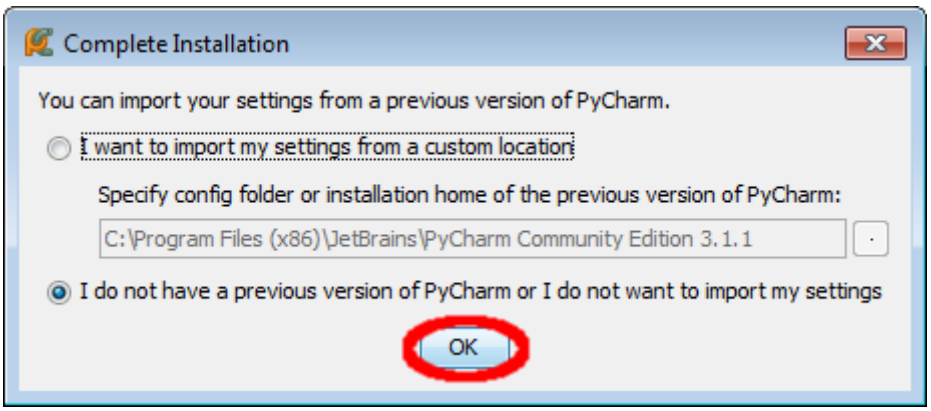
Cliquez sur “Install” pour installer le logiciel.



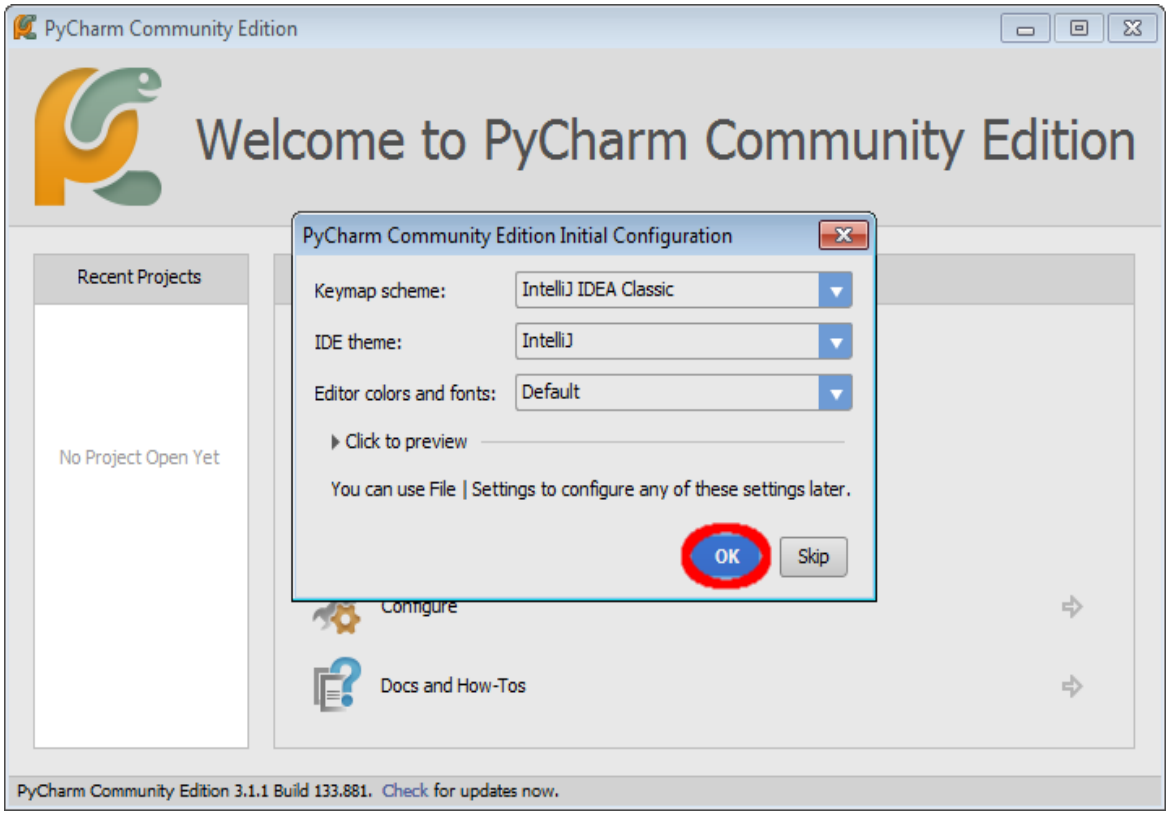
Cliquez sur “Finish” pour terminer l’installation.

Lancez PyCharm à l'aide du raccourci sur le bureau (ou par le menu Démarrer).

Comme il s'agit du premier lancement de PyCharm, nous allons procéder à sa configuration.

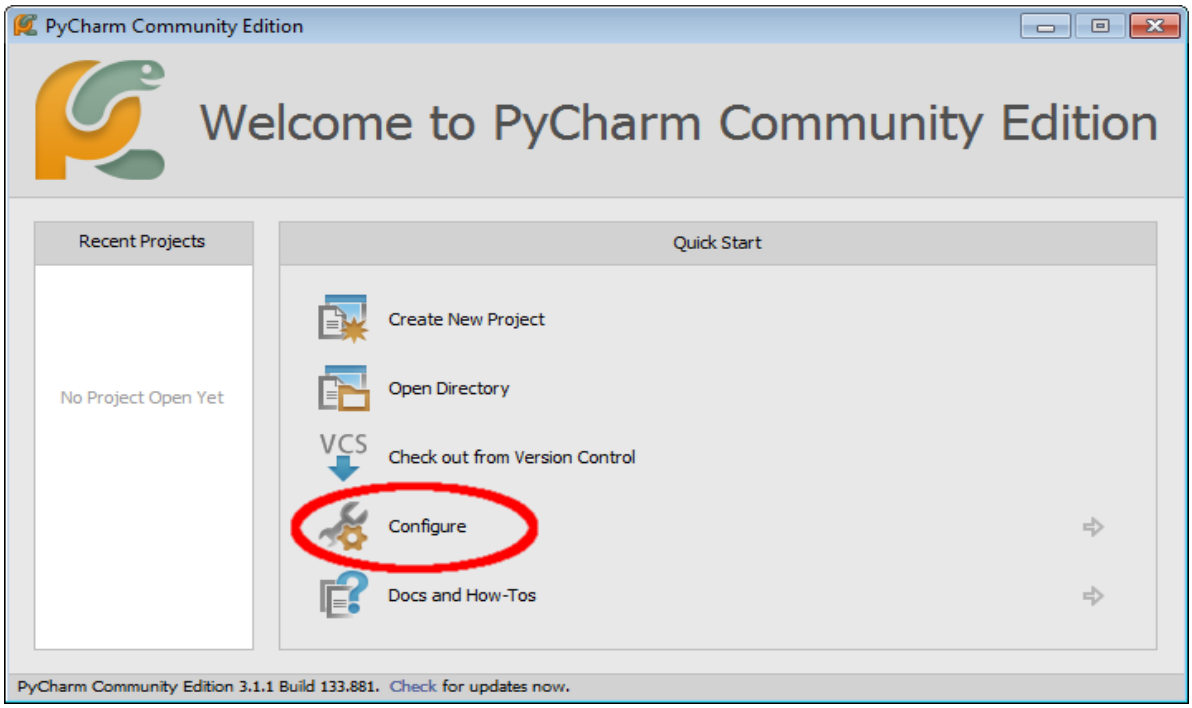


Cliquez sur "OK".

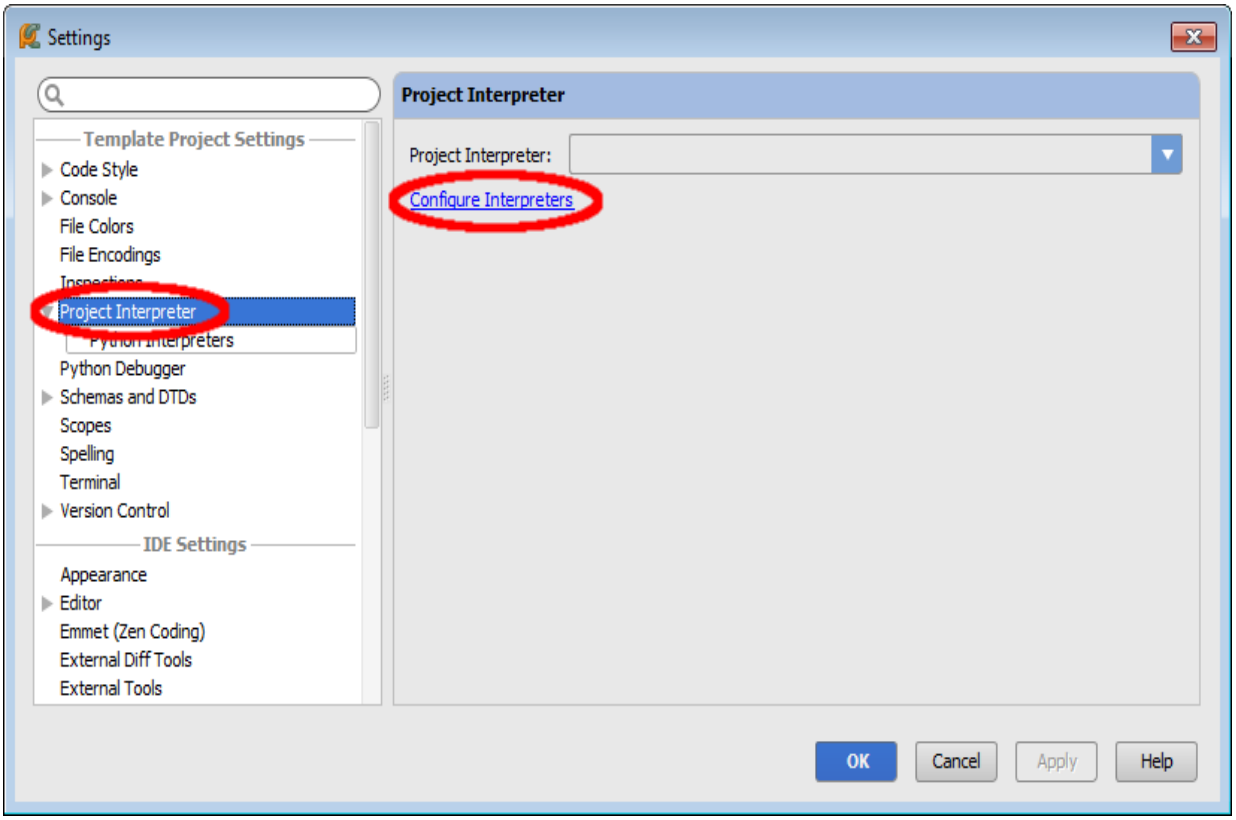


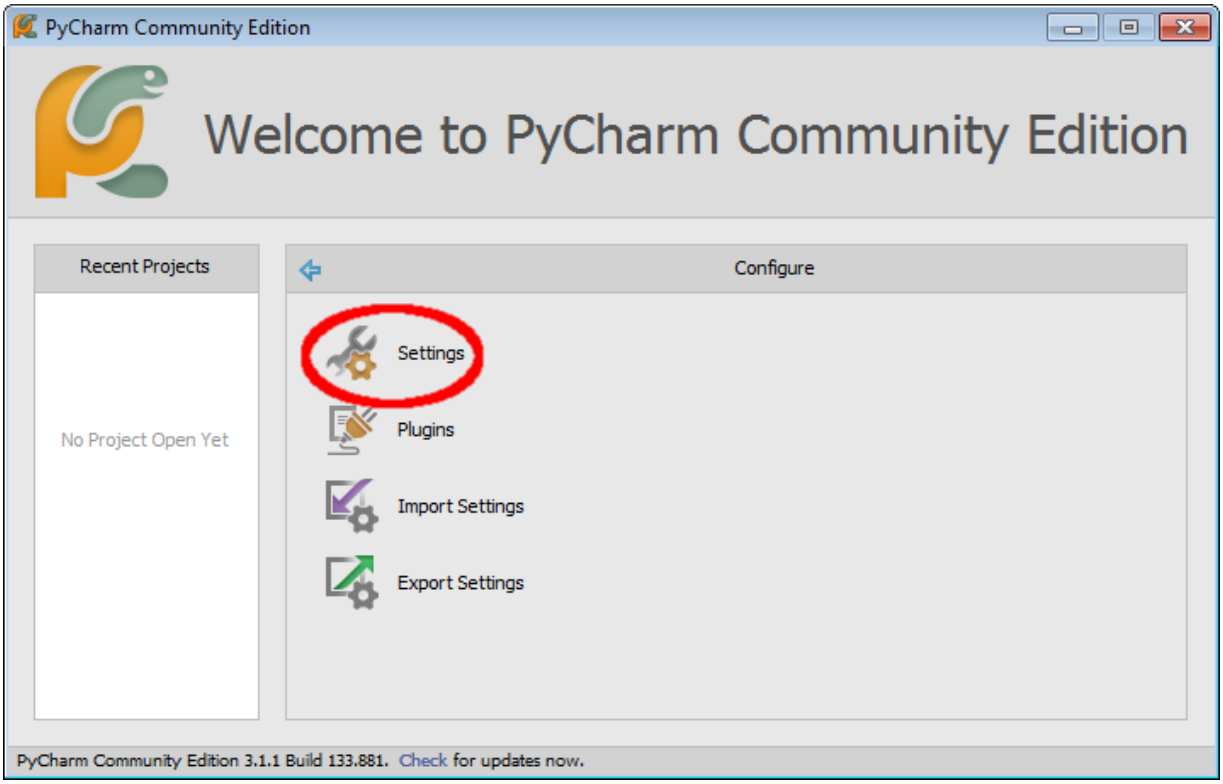
Cliquez sur "OK".

Cliquez sur “Configure”.

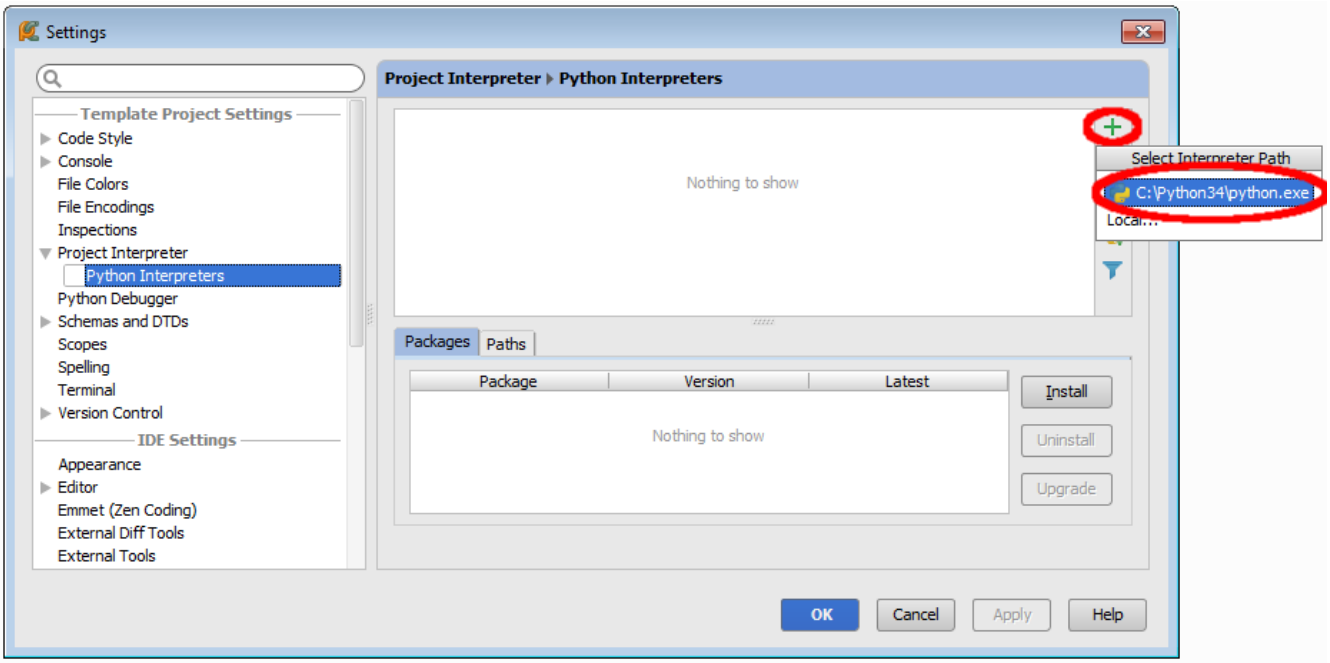


Cliquez sur “Settings”.

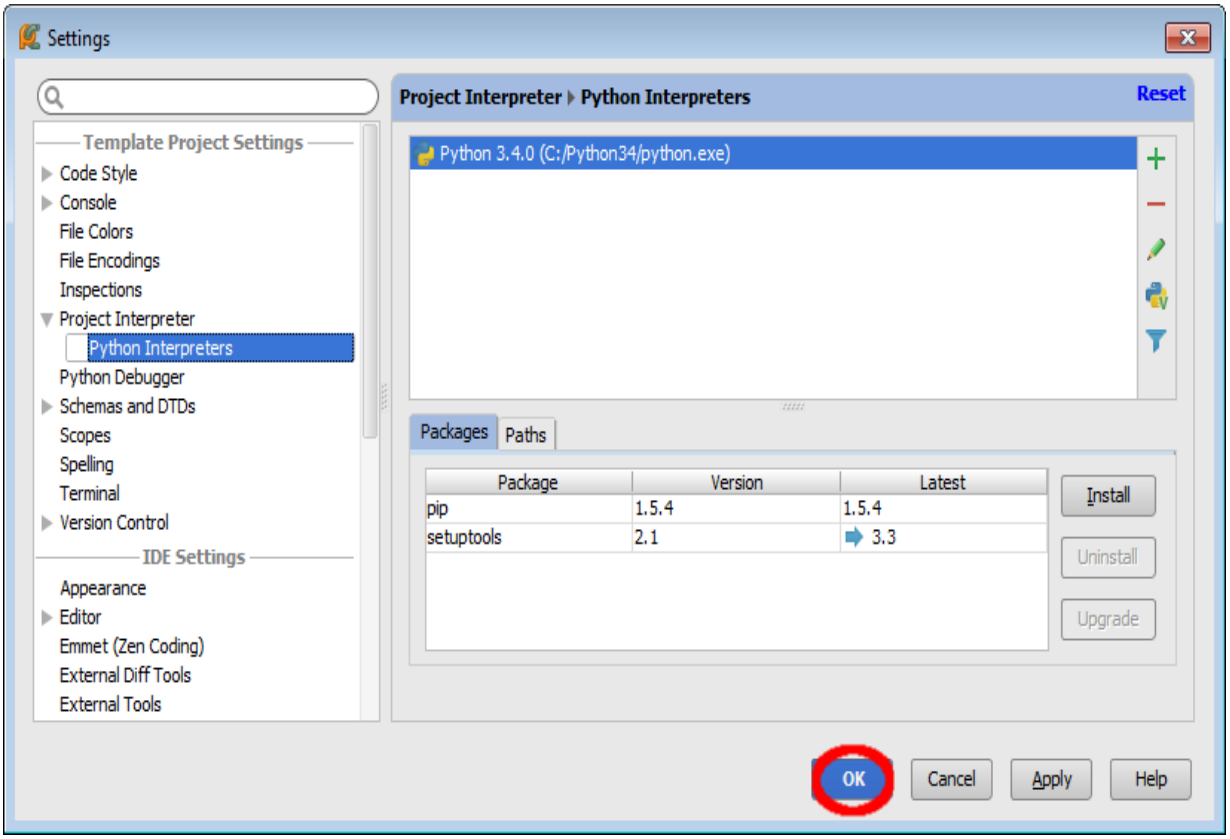




Cliquez sur “Project Interpreter” puis sur “Configure Interpreters”.



Cliquez sur le “+” à droite, puis sélectionnez l’interpréteur Python à utiliser.



L'interpréteur est à présent connu de l'environnement de développement.

Cliquez sur "OK".

Cliquez sur la flèche pour revenir à l'écran d'accueil.

III. Explication du code :

a. Fichier « crypto » :

Le ficher contient les fonctions intermédiaires, et les fonctions de cryptages/décryptages. On a commenté ces dernières pour vous expliquer comment elles fonctionnent. Prenons dans cette partie les définitions les plus intéressantes et sur lesquelles se base notre projet et qui sont celles de FEISTEL, RSA et les générateurs de clés de l'expéditeur A et du destinataire B :

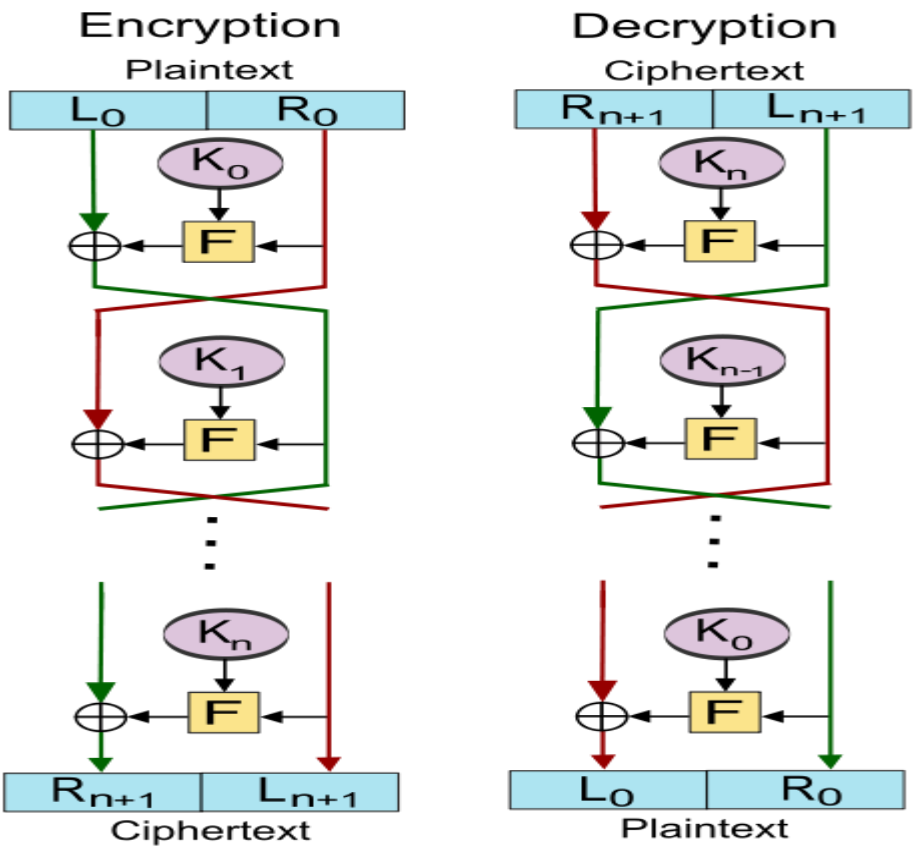
- **Fonction de cryptage FEISTEL :**

```
def encrypt_Feistel(Msg:str, Key:str):
    k1, k2, k3, k4 = Key[:2], Key[1:3], Key[2:], Key[3] + Key[0]
    g0, d0 = Msg[:2], Msg[2:]
    bin1, bin2 = [0 * w for w in range(5)], [0 * w for w in range(5)]
    bin = [0 * w for w in range(10)]


    #Tour 1
    x1 = d0
    y1 = g0
    bin1 = encrypt_2_char(y1)
    bin2 = function(x1,k1)
    for i in range (10):
        bin[i] = bin1[i] ^ bin2[i]

    #Tour 2
    x2 = decrypt_list_bin(bin)
    y2 = x1
    bin1 = encrypt_2_char(y2)
    bin2 = function(x2,k2)
    for j in range (10):
        bin[j] = bin1[j] ^ bin2[j]

    #Tour 3
    x3 = decrypt_list_bin(bin)
    y3 = x2
    bin1 = encrypt_2_char(y3)
    bin2 = function(x3,k3)
    for k in range (10):
        bin[k] = bin1[k] ^ bin2[k]

    #Tour 4
    x4 = decrypt_list_bin(bin)
    y4 = x3
    bin1 = encrypt_2_char(y4)
    bin2 = function(x4,k4)
    for l in range (10):
        bin[l] = bin1[l] ^ bin2[l]
    x5 = decrypt_list_bin(bin)

    return x4+x5
```

- **Fonction de décryptage FEISTEL :**

```
def decrypt_Feistel(Msg:str, Key:str):
    k4, k3, k2, k1 = Key[:2], Key[1:3], Key[2:], Key[3] + Key[0]
    g0, d0 = Msg[:2], Msg[2:]
    bin1, bin2 = [0 * w for w in range(5)], [0 * w for w in range(5)]
    bin = [0 * w for w in range(10)]


    #Tour 1
    y1 = d0
    x1 = g0
    bin1 = encrypt_2_char(y1)
    bin2 = function(x1,k1)
    for i in range (10):
        bin[i] = bin1[i] ^ bin2[i]

    #Tour 2
    x2 = decrypt_list_bin(bin)
    y2 = x1
    bin1 = encrypt_2_char(y2)
    bin2 = function(x2,k2)
    for j in range (10):
        bin[j] = bin1[j] ^ bin2[j]

    #Tour 3
    x3 = decrypt_list_bin(bin)
    y3 = x2
    bin1 = encrypt_2_char(y3)
    bin2 = function(x3,k3)
    for k in range (10):
        bin[k] = bin1[k] ^ bin2[k]

    #Tour 4
    x4 = decrypt_list_bin(bin)
    y4 = x3
    bin1 = encrypt_2_char(y4)
    bin2 = function(x4,k4)
    for l in range (10):
        bin[l] = bin1[l] ^ bin2[l]
    x5 = decrypt_list_bin(bin)

    return x5+x4
```

Fonctionnement :



Exemple :

```
print("Le message AAAA crypte avec la cle KXCX avec Feistel est : "
,encrypt_Feistel("AAAA","KXCX"))

print("Le message MYMW decrypte avec la cle KXCX avec Feistel est : "
,decrypt_Feistel("MYMW","KXCX"))
```

Le résultat est le suivant :

```
C:\Users\adminuser\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/adminuser/PycharmProjects/MI44FINAL/crypto.py
Le message AAAA crypte avec la cle KXCX avec Feistel est :  MYMW
Le message MYMW decrypte avec la cle KXCX avec Feistel est :  AAAA
```

On a vérifié à la main, et cela a donné le même résultat trouvé :

Exemple

msg : AAAA Key : KXCX

Tour 1 :

$g_1 = d_0 = AA$

$d_1 = g_0 \oplus \text{function}(d_0, KX)$ avec $g_0 = AA$

* function (d_0, KX) :

$AA = [00000, 00000]$

$KX = [01010, 10111]$

⚠ décalage de AA à gauche reste le même

$\text{XOR}(\overset{\text{décalage}}{AA}, KX) = [01010, 10111]$

$= KX$

alors

$d_1 = AA \oplus KX = KX$

Tour 2 :

$g_2 = d_1 = KX$

$d_2 = g_1 \oplus \text{function}(d_1, XC)$ avec $g_1 = AA$

* function (d_1, XC)

$KX = [01010, 10111]$

décalage $= [10101, 01110]$

$XC = [10111, 00010]$

$\text{XOR}(\overset{\text{décalage}}{KX}, XC) = [00010, 01100]$

$= CM$

alors

$d_2 = AA \oplus CM = CM$

Tour 3 :

$g_3 = d_2 = CM$

$d_3 = g_2 \oplus \text{function}(d_2, CX)$ avec $g_2 = KX$

* function (d_2, CX) :

$CM = [00010, 01100]$

décalage $= [00100, 11000]$

$CX = [00010, 10111]$

$\text{XOR}(\overset{\text{décalage}}{CM}, CX) = [00110, 01111] = GP$

$d_3 = KX \oplus GP$

$KX = [01010, 10111]$

$\oplus$

$GP = [00110, 01111]$

$\overline{d_3} = [01100, 11000]$

$= MY$

Tour 4 :

$g_4 = d_3 = MY$

$d_4 = g_3 \oplus \text{function}(d_3, XK)$ avec $g_3 = CM$

* function (d_3, XK) :

$MY = [01100, 11000]$

décalage $= [11001, 10000]$

$XK = [10111, 01010]$

$\text{XOR}(\overset{\text{décalage}}{MY}, XK) = [01111, 11010]$

$= [0, \ ]$

$d_4 = CM \oplus [0, \ ]$

$= [01100, 10110]$

$= MW$

Alors résultat final est :

MY MW

- Fonction de cryptage RSA

```
def encryptRSA(m : str, n : int, e : int):
    M = str2dec(m)
    C = puiss_mod(M, e, n)
    return C
```

- Fonction de décryptage RSA

```
def decryptRSA(C : list, n : int, d : int):
    D = puiss_mod(C, d, n)
    M = ""
    for i in range(4):
        M += dico[D[i]]
    return M
```

Ces deux fonctions utilisent deux définitions intermédiaires :

- Puiss_mod (string, int, int) : Qui permet de calculer l'exponentiation modulaire (m puissance e mod (n)), vu en cours

```
def puiss_mod(m, e, n):
    """ m est une liste d'entiers contenant des indices
de caracteres contenu dans le dico
        e cle publique
        n= p * q
    """
    indice = 0
    r = [0,0]
    res = [0*i for i in range(4)]
    for i in range(4):
        dec = e
        r[0] = 1
        r[1] = m[indice]
        while dec > 0 :
            if dec % 2 == 1 :
                r[0] = (r[0] * r[1]) % n
            dec = dec // 2
            r[1] = (r[1] * r[1]) % n
        res[indice] = r[0]
        if indice <= 3 : indice = indice + 1
    return res
```

- Str2dec (string) : Permet de transformer une chaine de caractère en une liste d'index de chaque lettre définie dans la liste proposée dans le TP1

```
def str2dec(s : str):
    l=[0*i for i in range(4)]
    for i in range(4):
        l[i] = find_dec(s[i])
    return l
```

Principe RSA:

¤ A crypte le message en utilisant la clé publique de B :

C = (M puissance eb) mod (nb)

¤ B reçoit le message et le décrypte le message en utilisant sa clé privée :

M = (C puissance db) mod (nb)

Le résultat trouvé après le test est comme suit :

```
********************  Cryptage du message d'identification par A   ********************
Le message que A veut envoyer a B est : AB?!
Le message crypte par A est : AB?R


********************  Decryptage du message d'identification par B   ********************
Le message decrypte par B est : AB?!
```

- Fonction de génération de mot de passe :

```
def password_generator(key:str):
    bin1 = encrypt_2_char("AB")
    bin2 = encrypt_2_char("OK")
    binkey1 = encrypt_2_char(key[:2])
    binkey2 = encrypt_2_char(key[2:])
    l1 = [0*i for i in range (10)]
    l2 = [0 * i for i in range(10)]
    for i in range (len(bin1)):
        l1[i] = bin1[i] ^ binkey1[i]
    for i in range (len(bin2)):
        l2[i] = bin2[i] ^ binkey2[i]
    return decrypt_list_bin(l1)+decrypt_list_bin(l2)
```

C'est une fonction qui permet de générer un mot de passe, en utilisant l'opérateur XOR (OU Exclusif) entre le message de confirmation « ABOK » et une chaine de caractères de 4 lettres générée aléatoirement par la fonction suivant :

```
#fonction qui genere une cle de 4 caracteres sous de string
def key_generator():
    res = ''
    for i in range (4) :
        res += dico[rand_num()%32]
    return res

#genere un nombre aleatoire modulo 32
def rand_num():
    return random.randint(0,1000) % 32
```

b. Fichier « C » :

Dans ce fichier, on génère les clés de l'expéditeur A et du destinataire B dans des fichiers texte (.txt) qu'on a nommé « donneesA.txt » et « donneesB.txt ». Le code est le suivant :

```
from crypto import *

#C genere la cle privee et publique de A
pa = randomPrime()
qa = randomPrime()
na = pa*qa
ea = e_generator(pa,qa)
da = d_generator(pa,qa,ea)

#C genere la cle privee et publique de B
pb = randomPrime()
qb = randomPrime()
nb = pb*qb
eb = e_generator(pb,qb)
db = d_generator(pb,qb,eb)

fa = open('donneesA.txt', 'w')
fa = open('donneesA.txt', 'a')
fa.write(str(pa) + "\n")
fa.write(str(qa) + "\n")
fa.write(str(na) + "\n")
fa.write(str(ea) + "\n")
fa.write(str(da) + "\n")
fa.close()

fb = open('donneesB.txt', 'w')
fb = open('donneesB.txt', 'a')
fb.write(str(pb) + "\n")
fb.write(str(qb) + "\n")
fb.write(str(nb) + "\n")
fb.write(str(eb) + "\n")
fb.write(str(db) + "\n")
fb.close()
```

Explication :

- **fa = open ('donneesA.txt', 'w') et fb = open ('donneesB.txt', 'w')**

'w' : ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé. Du coup à chaque fois qu'on va compiler le fichier « C », le fichier existant sera remplacé par un nouveau qui contient d'autres clés générées, qui seront écrites dans un ordre précis :

1ère ligne : **p** de A/B : nombre premier aléatoire de A/B
2ème ligne : **q** de A/B : nombre premier aléatoire de A/B
3ème ligne : **n** = p * q
4ème ligne : **e** de A/B : clé publique généré de A/B
5ème ligne : **d** de A/B : clé privée généré de A/B

- **fa.write(string)** : Pour écrire une chaine de caractères dans le fichier
- **fa.close()** : permet de fermer le fichier après l'avoir ouvert (importante)

c. Fichier « A » et « B » :

Dans ces fichiers, on trouve le code suivant :

```
import linecache

#recuperation des donnees necessaires pour B
pb = int(linecache.getline('donneesB.txt', 1))
pb = int(pb)
qb = int(linecache.getline('donneesB.txt', 2))
qb = int(qb)
nb = int(linecache.getline('donneesB.txt', 3))
nb = int(nb)
eb = linecache.getline('donneesB.txt', 4)
eb = int(eb)
db = linecache.getline('donneesB.txt', 5)
db = int(db)
na = int(linecache.getline('donneesA.txt', 3))
na = int(na)
ea = linecache.getline('donneesA.txt', 4)
ea = int(ea)
```

- **Linecache.getline('donnees « X ».txt', int):**

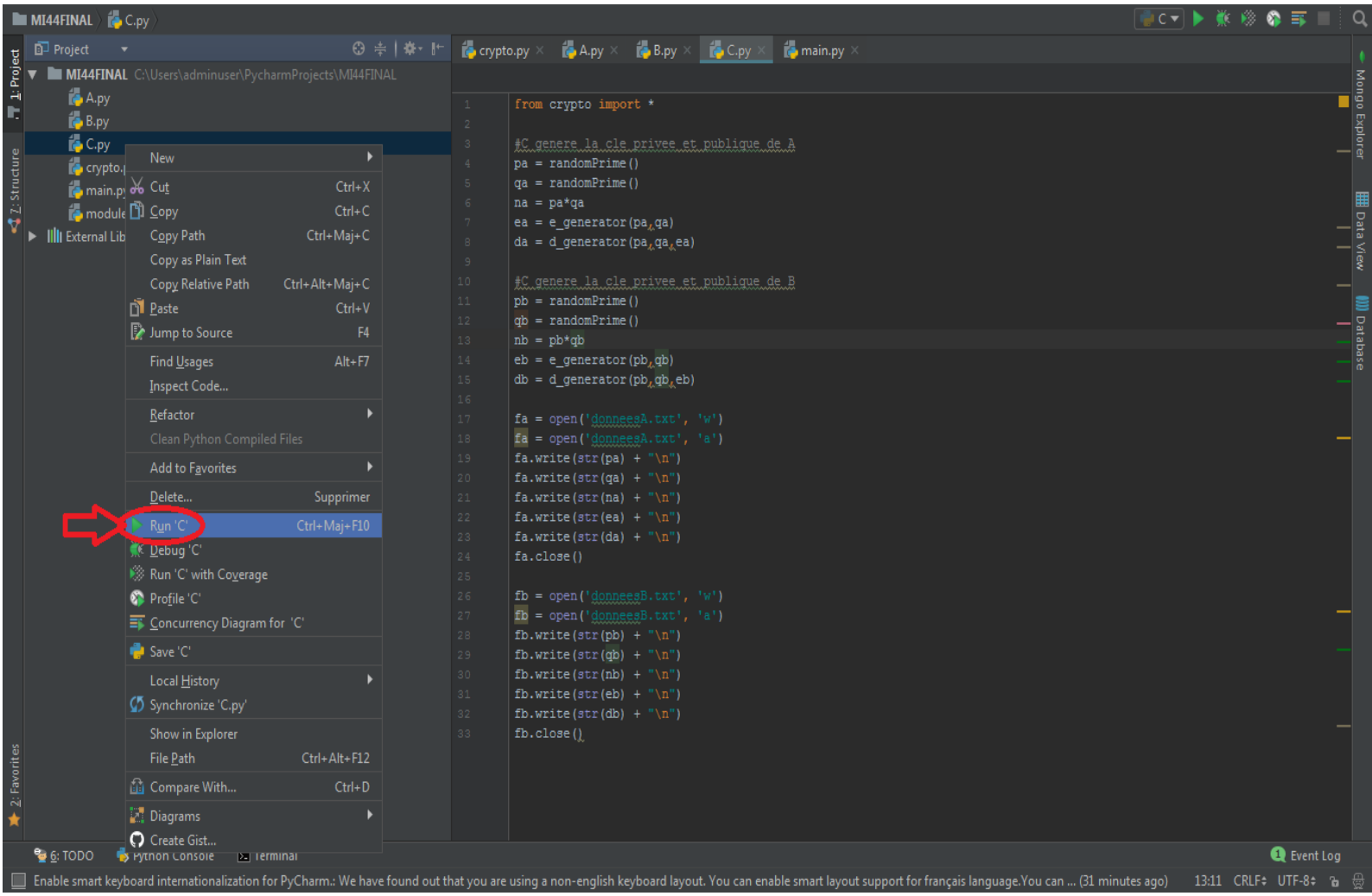
  Cette syntaxe ne fait que récupérer les données nécessaires à partir des fichiers texte généré par le programme « C », en choisissant les numéros des lignes voulues.

IV. Exécution :

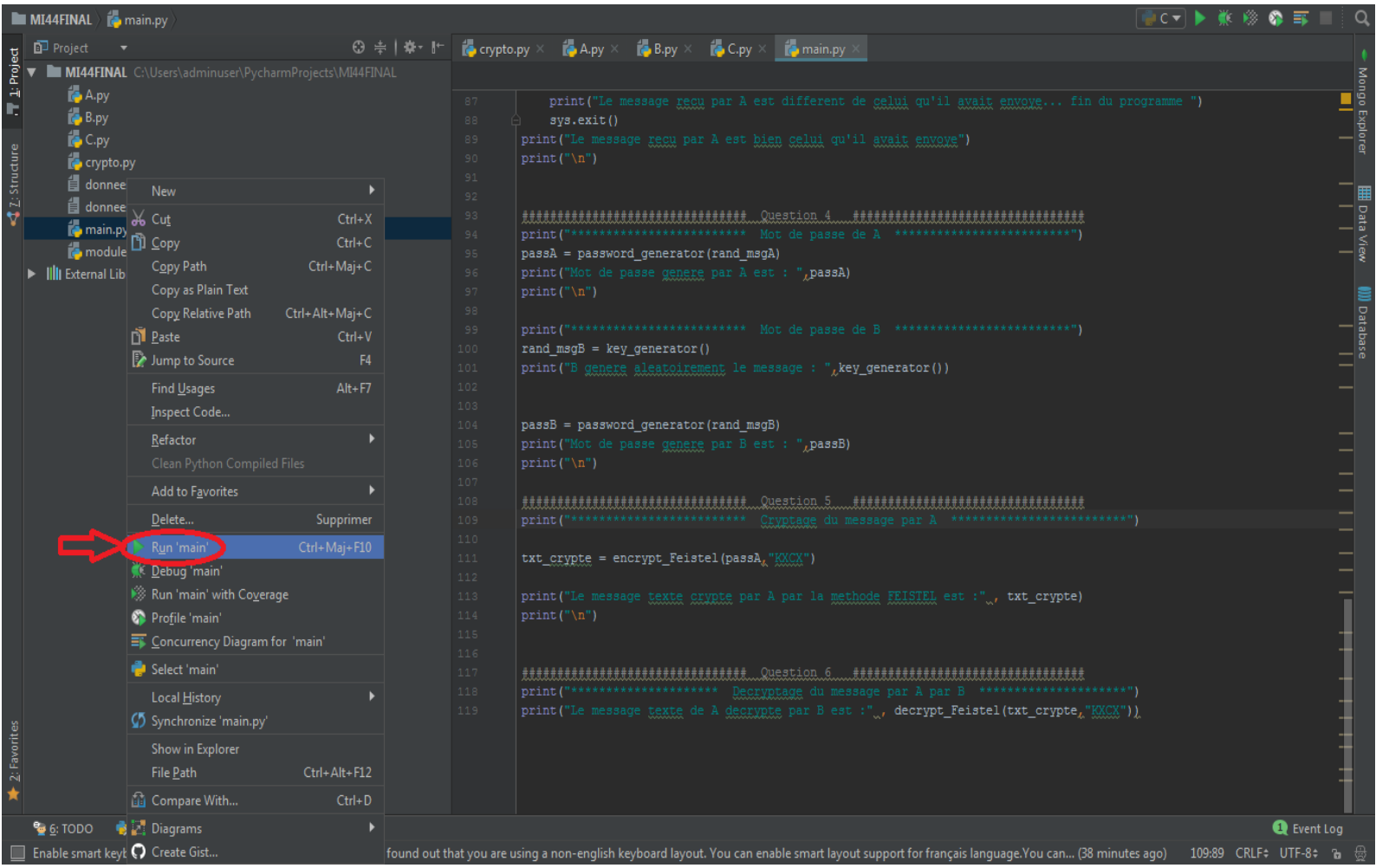
a- Ordre d'exécution :

On commence par compiler le fichier « C » dans l'environnement « PyCharm » avant le fichier « main », car, comme avait vu dans la partie [III. b.], il doit générer les deux fichiers textes qui contiennent les données de A et B.

¤ Run 'C' :



¤ Run ‘main’ :


MI44FINAL
main.py
Project
crypto.py
A.py
B.py
C.py
main.py
MI44FINAL C:\Users\adminuser\PycharmProjects\MI44FINAL
A.py
B.py
C.py
crypto.py
donnee
donnee
main.py
module
External Lib
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Maj+C
Copy as Plain Text
Copy Relative Path Ctrl+Alt+Maj+C
Paste Ctrl+V
Jump to Source F4
Find Usages Alt+F7
Inspect Code...
Refactor
Clean Python Compiled Files
Add to Favorites
Delete... Supprimer
Run 'main' Ctrl+Maj+F10
Debug 'main'
Run 'main' with Coverage
Profile 'main'
Concurrency Diagram for 'main'
Select 'main'
Local History
Synchronize 'main.py'
Show in Explorer
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Diagrams
Create Gist...
print("Le message recu par A est different de celui qu'il avait envoye... fin du programme ")
sys.exit()
print("Le message recu par A est bien celui qu'il avait envoye")
print("\n")
################################ Question 4 ################################
print("************************ Mot de passe de A ************************")
passA = password_generator(rand_msgA)
print("Mot de passe genere par A est : ",passA)
print("\n")
print("************************ Mot de passe de B ************************")
rand_msgB = key_generator()
print("B genere aleatoirement le message : ",key_generator())
passB = password_generator(rand_msgB)
print("Mot de passe genere par B est : ",passB)
print("\n")
################################ Question 5 ################################
print("************************ Cryptage du message par A ************************")
txt_crypte = encrypt_Feistel(passA,"KXCX")
print("Le message texte crypte par A par la methode FEISTEL est :", txt_crypte)
print("\n")
################################ Question 6 ################################
print("******************** Decryptage du message par A par B ********************")
print("Le message texte de A decrypte par B est :", decrypt_Feistel(txt_crypte,"KXCX"))
6: TODO
Event Log
Enable smart keyt
found out that you are using a non-english keyboard layout. You can enable smart layout support for français language.You can... (38 minutes ago)
109:89 CRLF UTF-8

b- Résultat :

Après avoir compiler le 'main' , on a eu un résultat comme ci-dessous et qui n'est qu'un exemple, car les résultats différent après chaque compilation de 'C' et 'main'.

Resultat

```
C:\Users\adminuser\AppData\Local\Programs\Python\Python36-32\python.exe C:/Users/adminuser/PycharmProjects/MI44FINAL/main.py


******************  Cryptage du message d'identification par A   ********************
Le message que A veut envoyer a B est : AB?!
Le message crypte par A est : ABTW


******************  Decryptage du message d'identification par B   ********************
Le message decrypte par B est : AB?!


******************  Cryptage du message de confirmation par B   ********************
Le message que B veut envoyer a A est : ABOK
Le message crypte par B est : ABKC


******************  Decryptage du message de confirmation par A   ********************
Le message decrypte par A est : ABOK


******************  Cryptage du message de A par A   ********************
A genere aleatoirement le message : QF,F
Le cryptage du message genere aleatoirement par A est : SCUC


******************  Decryptage du message de A par B   ********************
Le decryptage par B du message aleatoire envoye par A est : QF,F


******************  Cryptage du message de A par B   ********************
Le message recu crypte par B avec la cle publique de A est : AXEX


******************  Decryptage et verification de la reception par A  ********************
Le message decrypte par A pour verifier que B a bien recu son message precedent est : QF,F
Le message recu par A est bien celui qu'il avait envoye


************************  Mot de passe de A  ************************
Mot de passe genere par A est :  QESP


************************  Mot de passe de B  ************************
B genere aleatoirement le message :  F!,U
Mot de passe genere par B est :  QPFO


************************  Cryptage du message par A  ************************
Le message texte crypte par A par la methode FEISTEL est : PSR


```

Page 1 of 2

```
53 ********************  Decryptage du message par A par B  ********************
54 Le message texte de A decrypte par B est : QESP
55
56 Process finished with exit code 0
57
```