مدخل إلى لغة بايثون (Python)

لغة بايثون Python:

لغة بايثون (Python) هي لغة برمجة عالية المستوى (أنشئت سنة 1991 من طرف المبرمج الهولندي Guido van Rossum) تُستعمل لكتابة البرامج والتطبيقات بطريقة سهلة وواضحة. ميزتها أنها قوية وسهلة التعلم وتُعد من أكثر اللغات استعمالًا في العالم اليوم (نظرا لاحتوائها على الكثير من المكتبات الجاهزة، ولأنها مناسبة لمجالات عديدة وبالأخص الذكاء الاصطناعي).

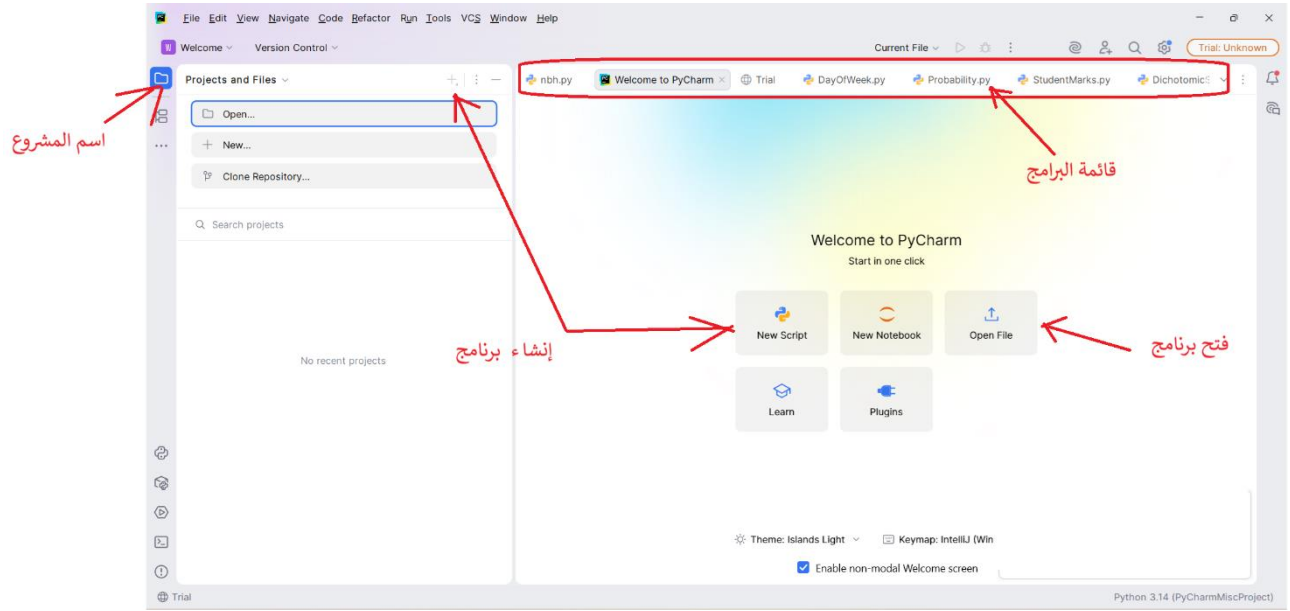
بيئة التطوير المتكاملة (IDE : Integrated Development Environment)أو(EDI بالفرنسية):

هي برنامج يسمح بـ كتابة وتشغيل واختبار وتصحيح البرامج المكتوبة في لغة ما داخل واجهة واحدة. يوجد الكثير من هذه البيئات المتخصصة في تطوير البرامج المكتوبة في لغة بايثون، منها: ***Pycharm*** ،***Visual Studio Code*** ، ***Spyder*** . كما توجد الكثير من البيئات تعمل على الانترنت منها : *Google Colab* ، *Replit* ، ***Kaggle Notebooks***.

في حصص الأعمال التطبيقية نستعمل البرنامج *Pycharm*، أو البيئة : "*https://www.online-python.com*/".

البرنامج Pycharm:

عند فتح البرنامج تظهر النافذة التالية



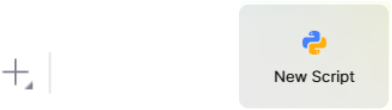
إنشاء مشروع جديد:

المشروع يتكون من مجموعة من البرامج.

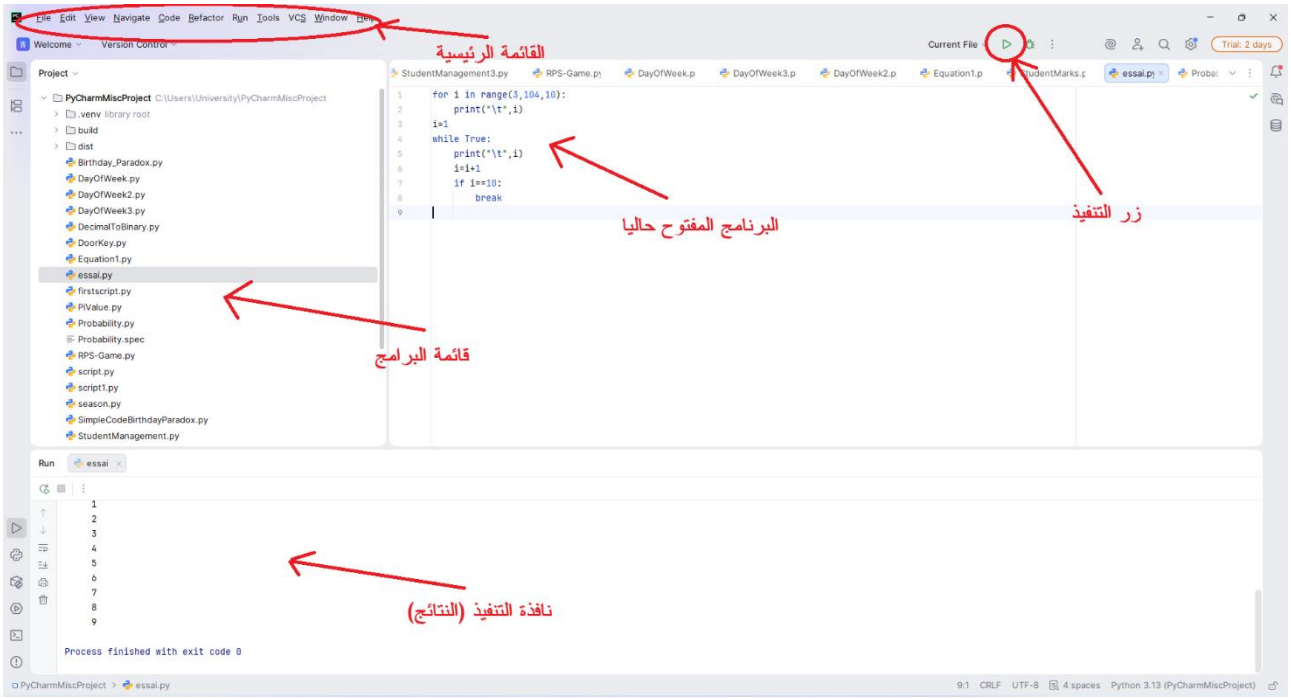
لإنشاء مشروع جديد يتم بالنقر على "New" -- > "New Project"

إنشاء برنامج جديد :

النقر على الزر المسمى "New Script" أو على الزر الموسوم بالإشارة "+" يقوم بإضافة برنامج جديد (فارغ) إلى المشروع الحالي.



عند اختيار برنامج تظهر الواجهة:



حفظ برنامج:

بالضغط على الزرين : ctrl + s معا، أو عن طريق تنفيذ البرنامج فيتم حفظه بطريقة تلقائية.

أساسيات لغة بايثون:

- لا يوجد تصريح مسبق للمتغيرات كما في بعض اللغات مثل C أوJava أو في الخوارزميات، المتغير يُنشأ بمجرد إعطائه قيمة.
مثل : التعليمة A=3 تنشئ متغيرة A من نوع عدد صحيح وبه قيمة 3.

```
x = 10              # عدد صحيح
y = 3.14            # عدد عشري
name = "Ali"        # نص
is_ok = True        # منطقي
```

- **المسافة البادئة (التسنين) (Indentation):** باستعمال المسافات، كل تعليمة في سطر جديد، التعليمات التي تنتمي إلى نفس التركيبة تكون في نفس العمود متقدمة بمسافة عن بداية التركيبة. نهاية التركيبة تكون عند أول تعليمة تبدأ في نفس العمود مع بدايتها.

مثال:

التعليمتين (m=m+12) و(y-=1) تابعتين للتركيبة (if)
أما التعليمة (k=y%100) فهي خارجة عنها.

```
if m<=2:
    m = m + 12
    y -= 1
k = y % 100
```

- **التعليق (comment):** باستعمال الرمز #.
- **تعليمة الإسناد:** هي " = " مثل : m = m +12 . للتعبير عن المساواة نستعمل " == " وبـ " != " عن عدم المساواة.
- **تعليمة القراءة (input) :** تعطي دائما نتيجة من نمط "سلسلة حروف".مثال

A : من نوع سلسلة حروف
B: عدد صحيحي (إضافة int للتحويل)
C: عدد حقيقي (إضافة float للتحويل)
لاحظ أن تعليمة القراءة تمكن من إظهار نص للمستخدم
(من أجل تنبيهه على ضرورة إدخال قيمة)

```
A=input("Enter the value of A: ")
B=int(input("Enter the value of B: "))
C=float(input("Enter the value of C: "))
```

- **تعليمة الكتابة (print):** مثال :

```
print("\n\tThe solution is : ",X)
```

  - وضع الرمز \n داخل السلسلة النصية يؤدي إلى القفز إلى السطر الموالي
  - وضع الرمز \t داخل السلسلة النصية يؤدي إلى ترك مسافة فارغة قبل كتابة السلسلة.

- **العمليات الحسابية :** الجمع "+"، الطرح "-"، الضرب "*"، القسمة الحقيقية "/"، القسمة الصحيحة "//"، باقي القسمة "%"، الأس "**" (2**3 معناها 3 أس 2).

التركيبات:

- تركيبات الاختيار :

```
if h == 0:
    print("\tSaturday")
elif h == 1:
    print("\tSunday")
elif h == 2:
    print("\tMonday")
else:
    print("\t\tFalse value")
```

```
if m<=2:
      m = m + 12
      y -= 1
else:
      m = m + 1
k = y % 100
```

```
if m<=2 :
      m = m + 12
      y -= 1
k = y % 100
```

– لاحظ وجود النقطتين عند نهاية سطر (if) (else) (elif).

التركيبة "حسب" (switch).
لاحظ استعمال (_ case) للدلالة على الحالات الأخرى (else).

```
match h:
        case 0: Day="Saturday"
        case 1: Day="Sunday"
        case 2: Day="Monday"
        case 3: Day="Tuesday"
        case 4: Day="Wednesday"
        case 5: Day="Thursday"
        case 6: Day="Friday"
        case _: Day="False value"
print("\n\tThis week's day is ",Day)
```

- الحلقات:

استعمال كلمة (range) للدلالة على المجال من 0 إلى n. في هذه الحالة i تأخذ القيم من 0 إلى n-1.
يمكن كتابتها على الشكل for i in range(n) .
الكتابة for i in range (1,10,3) تعبر عن القيم من 1 إلى 9 باستعمال الخطوة (step) 3.

```
for i in range(0,n):
    print("\t",Courses[i],":",Marks[j][i])
```

الصيغة (while True)تعبر عن التركيبة "كرر" (repeat) .
لاحظ ضرورة استعمال التعليمة break للخروج من هذه التركيبة عندما يصل i إلى 10.

```
while True:
    print("\t",i)
    i=i+1
    if i==10
        break
```

```
while i<n:
      print("\t",i)
      i=i+1
```

- **تركيبة معالجة الأخطاء "حاول" (try):**

هي تركيبة تستعمل لمعالجة الأخطاء التي يمكن أن يقع فيها المستخدم عندما يريد إدخال قيمة متغيرة (مثلا المطلوب إدخال عدد ولكن المستخدم يدخل حرف).

```
try:
    x = int(input("Enter a number: "))
    print(10 / x)
except:
    print("try again please")
```

- إذا أدخل المستخدم قيمة صحيحة فإن التعليمات الموجودة في الجزء "try" هي التي تنفذ. أما إذا أخطأ فستنفذ التعليمة الموجودة في الجزء "except".

بعض هياكل البيانات في بايثون:

القائمة:

```
numbers = [1, 2, 3, 4]
print(numbers[0])  # الوصول لعنصر
numbers.append(5)  # إضافة عنصر
```

الجداول البسيطة:

إذا كان الجدول بسيطا (ذو بعد واحد) نستعمل القائمة.

مثال:

T = [10,20,30,40]

المصفوفات : المصفوفات هي عادة جداول ذات بعدين (أسطر وأعمدة):

- المؤشرات تبدأ من 0.
- المؤشر الأول يدل على رقم السطر والثاني على العمود:

mx[2,1] يدل على القيمة 8.

```
mx = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
 ]
```

مثال: قراءة عناصر مصفوفة متكونة من عدد من الأسطر وعدد من الأعمدة:

```
Row=int(input("Number of rows :"))
Col=int(input("Number of columns:"))
M=[]
M=[[0 for j in range(Col)] for i in range(Row)]
for i in range(Row):
    for j in range(Col):
        M[i][j]=int(input("M["+str(i)+","+str(j)+"]="))
print(M)
```

الإجراءات والدوال:

لغة بايثون لا تُميِّز بشكل صريح بين مفهومي الدالة (***Function***) والإجراء (***Procedure***) كما هو الحال في بعض اللغات التقليدية. فالدالة تستعمل التعليمة return لإرجاع قيمة بينما الإجراء لا يرجع أي قيمة.

نستعمل كلمة def للتصريح بالدوال في بايثون:

```
def Permut(X, Y) :
    Z=X
    X=Y
    Y=Z
    return X,Y
. . .
A,B = Permut(A,B)
```

```
def Welcome(name) :
    print(f"Say Welcome to  {name} !")
. . .
welcome("Nabil") # this is the call
```