PYTHON COURSE

Python course , from scratch to Professionalism ...

Made by : Taha Khaldoun Amro

Functions

لا نبالغ في الأمر حين نقول أن هذا هو أهم درس حتى الان , فمفهوم الfunctions يعد أحد أهم المفاهيم في عالم البرمجة كلها , وليس فقط في لغة بايثون .
يعبر هذا المفهوم عن قطعة من الكود , يمكن كتابتها و إعادة استخدامها أكثر من مرة في الكود , مما يقتصر علينا إعادة كتابتها مرات عديدة .
و الان بعض خصائص الfunctions التي سنتعرف عليها في أثناء عملنا على الموضوع :

1. الfunction يتم تنفيذه حين استدعائه فقط
2. الfunction قد تعمل على تنفيذ كود دون إرجاع معطيات
3. الfunction يأخذ قيمة لتنفيذ الكود عليها تسمى parameter
4. قد يعمل الfunction على إرجاع بيانات بعد القيام بعملية على بيانات أخرى معطاة له على شكل parameter
5. الfunction خلقت لمنع مبدأ ال(DRY (dont repeat yourself
6. الfunctions تقبل عناصر عندما يتم استدعاؤها تسمى الarguments
7. يوجد اقترانات مدمجة بالنظام و اقترانات يتم برمجتها من المستخدم
8. يمكن استدعاء الfunctions من ملفات أخرى ولا داعي لإعادة كتابتها في كل ملف

و الان و بعد أن تعرفنا على خصائص الfunctions , سنعمل الان على صنع واحد بشكل عملي .

Functions

طريقة تعريف الاقتران :
ببساطة , يتم تعريف الاقتران في لغة بايثون عن طريق الأمر def و الذي يوضع بعده اسم الاقتران , وبعد اسم الاقتران نضع قوسين , ومن ثم نضع نقطتين رأسيتين و نبدأ بكتابة كود الاقتران .
سنبدأ الان بتعريف اقتران للترحيب بالمستخدم لا أكثر , لنتعلم مبدأ العمل فقط :

```
def greeting() :
 print('Hello')

greeting()
```

كما رأينا , فإن مبدأ العمل بسيط جداً , فبعد تعريف الاقتران , تقوم باستدعائه متى ما أردت في داخل الكود , وهذا ما يجعل الاقتران ينفذ الكود الذي وضع داخله .
كنا قد ذكرنا سابقاً وجود نوع من أنواع الاقترانات التي لا تعمل على إرجاع قيمة , بل هي فط تقوم بتنفيذ أمر موجود داخلها , مثل هذا المثال , أنت لا تحصل على معلومة عند تنفيذك للاقتران , كل ما تحصل عليه هو تنفيذ للأمر الموجود داخل الاقتران .

Return functions

و فيما يخص الاقترانات التي تعمل على إعادة قيمة يمكن التعامل معها , فإنها ببساطة بدلاً من تنفيذ أمر , تعيد قيمة للمستخدم , و هو بإمكانه إجراء أي عملية يريدها عليها , حيث يمكن إعادة أي نوع من البيانات من الاقتران .

```
def return_me_a_value():
  return 100

print(return_me_a_value() + 100)
```

في الكود السابق , قمنا بتعريف اقتران يعيد لنا قيمة عددية (100) باستخدام الأمر return , و من ثم استدعينا الاقتران و قمنا بإضافة 100 إلى القيمة التي تم إرجاعها .
لاحظ في المثال السابق أنك لو قمت بإزالة أمر الطباعة , فلن يظهر لك شيء , لأن الاقتران لا ينفذ كوداً في الحقيقة , فقط يعمل على إرجاع قيمة و يمكنك أنت كمبرمج التعامل معها كما تشاء .
جرب الان التالي :

- خزن القيمة كمتغير
- غير القيمة المرجعة ألى نص و قم بتنسيقه باستخدام تقنيات النصوص .

Parameters and Arguments

لنقل أنك تعمل على كتابة نظام يعمل على القيام بعملية حسابية لشركة كبيرة , و خطرت على بالك فكرة تحويل العملية الى اقتران لتستخدمها كل أقسام الشركة , في العملية يتم إدخال قيمتين رقميتين , يطلب منك إرجاع قيمة تمثل : القيمة الأولى مرفوعة للقوة القيمة الثانية.

إذا فنحن نتعامل مع مدخلات , و لكن , كيف يمكننا إدخال هذه القيم إلى الاقتران ؟

ببساطة يكمن الحل في مفهوم الparameters و الarguments , و قبل تنفيذ النشاط سنتعرف عليهما :

```
def addition(first_num , second_num): # in the brakets are parameters

  print(first_num + second_num) # the sum of both parameters

addition(12 , 164) # arguments
```

في الكود السابق , قمنا بعملية جمع لعددين عن طريق تعريف اقتران يقبل قيمتين , الأولى تعبر عن الرقم الأول , و الثانية تعبر عن الرقم الثاني , وعند مناداة الاقتران قمنا بإدخال قيم العددين بالترتيب , فلو أدخلنا عدداً واحداً , فإننا سنواجه error يخبرنا بأن المطلوب عددان و ما تم إعطاؤه للاقتران هو عدد واحد فقط .

Parameters and Arguments

بالعودة إلى المهمة التي طلبت منا , فإننا ببساطة بيمكننا استخدام الاقتران من المثال السابق , و ذلك لأن المطلوب من المستخدم إدخاله هو عددان , و تبديل أمر الاقتران بعملية إرجاع لقيمة العدد الأول مرفوعاً لقوة العدد الثاني , استخدم الكود من المثال السابق و غير اسم الاقتران إلى شيء مناسب , و غير العملية التي يقوم بها الاقتران .

تكمن الخدعة هنا في عملية إدخال الرقمين , فقد طلب منا أن يتم ادخال المدخلات من المستخدم , حيث سنستخدم هاتين القيمتين كمدخلات للاقتران الخاص بنا , و يتم الأمر كالتالي :

```
firstnum = int(input('Enter first number : '))
secondnum = int(input('Enter second number : '))

print(calculate(firstnum , secondnum))
```

كل ما قمنا به ببساطة هو إدخال قيمتين من المستخدم , و إدخالهما في الاقتران , عند تنفيذ الكود , يدخل المستخدم الأرقام , يقوم الاقتران بعملية إرجاع القيمة بعد تنفيذ العملية على المدخلات .
هذا هو الأمر بكل بساطة .

Packing and Unpacking

في الأمثلة السابقة , تعلمنا كيفية إعطاء الfunction مجموعة من القيم حتى تتم عملية عليها , و تعرفنا على مفهوم الargumentsو الparameters , في هذا الدرس , سنتحدث عن مشكلة تواجه العديد من المبرمجين , يمكن تمثيلها بالمثال التالي :

تخيل أنك تعمل على إنشاء اقتران , يرحب بالمستخدمين الجدد لموقعك , حيث يطبع أسماءهم واحداً تلو الاخر دون النظر إلى عددهم , تكمن المشكلة هنا , لا يمكننا في أثناء إضافتنا للparameters في الfunction حصر عدد المستخدمين الجدد , فقد يرحب النظام ب7 مستخدمين , وقد يرحب بمستخدم واحد فقط , و هنا يكمن الحل في عملية الunpacking سنعمل الان بالطريقة التقليدية , و سنعتبر أن الحد الأقصى للمستخدمين الجدد هو 4 فقط :

```
def say_hello(n1 , n2 ,n3 ,n4):

  users = [n1 , n2 , n3 , n4]
  for user in users :
    print(f'Hello {user}')

say_hello('Osama' , 'Ahmed' , 'Samera' , 'Jomana')
```

Packing and Unpacking

لو جربت في الكود السابق إضافة شخص خامس للترحيب به , فإن العملية لن تتم و سيتم إعادة error يخبرك بأنك قد تخطيت الحد المسموح به ألا وهو 4 .
تأتي عملية الUnpacking و الPacking في هذا المثال للعمل على عدم حصر المستخدمين بعدد معين , حيث تمكننا من إدخال عدد غير نهائي من المستخدمين للترحيب بهم .
يتم ذلك من خلال تبديل الparameters الموجودة في تعريف الاقتران بمتغير واحد يعبر عن list تتكون من كل ما يتم إدخاله من المعلومات من قبل المستخدم , عن طريق وضع اسم المتغير الذي سنسميه بـusers بعد إشارة الضرب , والتي تعني أنه يتم تجميع بيانات في هذا المتغير , وإزالة الlist الخاصة بالمستخدمين في الكود السابق , و عمل loop على الparameter users .

```
def say_hello(*users):

  for user in users :
    print(f'Hello {user}')

say_hello('Osama' , 'Ahmed' , 'Samera' , 'Jomana' , 'Majed')
```

دورة تعليم لغة بايثون

KWargs

يعتبر مفهوم الkey word arguments من المفاهيم البرمجية المعقدة الخاصة بالfunctions , كما أنه ليس ضروريا في عملنا , لذلك فنحن لن نتكلم عنه , و لكن سنترك لكم رابطين لشرح العملية و كيف تتم و ما هي فائدتها مع تدريب عليها :

الروابط :

https://youtu.be/pMeKs94OrxQ?si=IyPWPmzrqA6E44O5

https://youtu.be/7o58LMti2po?si=8OKCNJk4nBil17WU

هذه الروابط للاطلاع فقط , حتى تتمكن من معرفتها إذا ما واجهتها مي مسيرتك المستقبلية .

Functions scope

سنتعرف الان إلى أحد أهم المفاهيم البرمجية التي يخطئ فيها الكثيرون , ألا و هي مبدأ الglobal و الlocal .
يعبر هذان المبدءان عن مفهوم الscope و هو عملية تخصيص المتغيرات و تحديد نطاق استعمالها .
في البداية , قم بوضع متغير x و أعطه قيمة رقمية 1 , ثم قم بإنشاء function يعمل على طباعته , ثم اطبعه مرة باستخدام الfunction و مرة بدون الfunction , لا يوجد فرق أليس كذلك ؟
يسمى المتغير الذي وضعناه الان بمتغير عالمي , أو global scope حيث يمكن لأي جزئية في الكود الاستفادة منه , يمكن تشبيه ذلك بالدولة التي تحتوي على مدن , حيث تعبر الfunctions عن المدن و الكود كله هو الدولة , جرب الان أن تغير قيمة x داخل الfunction و قم بتنفيذ الكود .
لا بد من أنك حصلت على نتائج مختلفة هذه المرة , فما حصل هنا أن الFunction لا يمكنها تغيير خارجها , ولا يمكنها التحكم في ما هو خارجها , حيث يمكننا شرح الأمر بالطريقة التالية :
عندما يكون لديك متغير عالمي , يمكن للfunctions الولوج إليه , و لكن عندما تحتوي الfunction ذاتها على نفس المتغير بقيمة جديدة , فهي بطبيعة الحال أصبحت مستغنية عن المتغير العالمي , و لكنها لا يمكنها أن تصدر القيمة التي أعطيت للمتغير إلى خارج الfunction , و هذا ما يسمى بالlocal scope .
يمكن تشبيه الأمر أيضا بالمثل الأمريكي الشهير : What happens in vegas , stays in vegas , و معناه أن ما يحدث داخل المدينة , يبقا داخلها , إلى أن يتم تصديره .

Functions scope

للتأكد فقط , يجب أن يكون الكود الان لديك كالتالي :

```
x = 1

def one() :
  x = 2

  print(f'this is x from local {x}')

print(f'this is x from global {x}')
one()
```

نستنتج الان أنه و مهما وضعت من اقترانات و غيرت داخلها قيمة x فإن قيمة x العالمية لن تتغير .
و لكن ماذا نفعل اذا أردنا أن نغير قيمة متغير عالمي من داخل اقتران ؟
يتم ذلك الأمر من خلال تعريف النظام بأن المتغير الذي سيكتب لاحقا سيكون global , حيث يجب كتابة أمر التصدير قبل تعريف المتغير داخل الاقتران هكذا :

Functions scope

```
x = 1

def one() :
  global x

  x = 2

  print(f'this is x from local {x}')

print(f'this is x from global {x}')
one()
```

الأمر الطريف هنا , أنك حتى لو قمت بتنفيذ الكود الان , فلن يتغير شيء على الإطلاق , لأنك فعلت الاقتران بعد أن طبعت المتغير للمرة الأولى , جرب الان تفعيل الاقتران قبل أن تطبعه و ستحصل على النتيجة التي نبحث عنها .
و هذا هو كل ما في الأمر يا سادة .

Recursion

سنتكلم الان عن اخر جزئية في عالم الfunctions , ألا و هي الrecursion و هي عملية إعادة استدعاء نفس الfunction داخل نفسه , حيث ينفذ الاقتران نفسه مراراً و تكراراً .
و لتتعرف على هذا المفهوم سنعمل على إنشاء نظام معقد بعض الشيء و لكننا لن نفهم الموضوع إلا من خلاله .
سنعمل الان على كتابة كود يعمل على إزالة الحروف المكررة من الكلمة و إعادتها بدون الحروف المكررة , فمثلاً لو أدخلنا الكلمة (wwwoooorrrldd) سيتم إرجاع الكلمة world .
في البداية لنتذكر عملية تقطيع النصوص , فلو مثلاً أردنا أن نحصل على الحرف الأول من الكلمة , نكتب الكود الاتي :

```
word = 'wwwoooorrrldd'
print(word[0])
```

و لو أردنا طباعة كل الكلمة من دون الحرف الأول ننفذ الكود :

```
print(word[1:])
```

ستكون هاتين العمليتين هما مبدأ عملنا , لذلك ابقهما موجودتين في دماغك .

Recursion

سنسمي الان اقترانا المطلوب , سنسميه clean_word , و نعطيه parameter (word) .
بعد ذلك سننحل الأمر بشكل منطقي , ماذا سيفعل الكود اذا كانت الكلمة مكونة من حرف واحد أصلا , بطبيعة الحال سوف يعيد لنا الكلمة نفسها دون تغييرات , و ذلك عبر الكود الاتي :

```
def clean_word(word) :

  if len(word) == 1:
    return word
```

حسنا , لنقل أن الكلمة أكثر من حرف واحد , ماذا سيفعل الكود , بشكل منطقي , سيعمل الكود على فحص ما اذا كان الحرف الأول مساويا للثاني الذي يليه , حيث أنه في حال كانا نفس الحرف , سوف تتم العملية على نفس الكلمة - الحرف الأول , و عندها سنستخدم عملية الSLICING التي ذكرناها سابقا , أضف الكود هذا داخل الfunction :

```
  if word[0] == word[1] :
    return clean_word(word[1:])
```

Recursion

دقيقة حتى نستوعب ما تم في الكود الاخير , كما قلنا فان النظام وجد أن الكلمة تتكون من أكثر من حرف واحد , لذا فان النظام يبحث ما اذا كان الحرف الأول مساويا للحرف الثاني , فلو وجدهما متساويين , يعيد النظام تنفيذ نفسه , على ما تبقا من الكلمة , أي الكلمة نفسها بدون الحرف الأول .
لو جربت الكود الان فلن تحصل على نتيجة , و ذلك لان العملية لن تتوقف , وعندما تتوقف لأن حرفا ما لا يساوي ما بعده , فلن يقوم النظام باعادة اي شيء و هذا ما سنعمل عليه الان .
في سبيل فهم ما حدث سابقا و الحصول على نتيجة , أضف جملة اعادة للكلمة في نهاية الاقتران : return word
عندما نفذت الكود , أعاد لك النظام الكلمة مع الغاء تكرار الحرف الأول فقط , ماذا عن باقي الحروف ؟
ببساطة , لم نقم نحن باخبار النظام ماذا يفعل اذا لم تكن الأحرف متساوية , حيث أنه في حال لم يكن الحرف الأول مساويا للحرف الثاني يجب عليه تخزين الحرف الأول و تنفيذ الامر على ما تبقى من الكلمة (الكلمة بدون الحرف الأول الذي خزنناه)
أضف الكود التالي الى الfunction و ستكون الأمور قد انتهت :

```
cleaned_word = word[0]
return cleaned_word + clean_word(word[1:])
```

ما حصل هنا هو أن العملية تمت مرات و مرات , وفي كل مرة يتم تخزين الحرف الأول الذي يكون مميزا .

Recursion

قد تبدو العملية الأخيرة التي قمنا بها معقدة بعض الشيء , و هي كذلك , لكنها الطريقة الوحيدة لفهم المضوع و ذلك لأهميته .
مرفق لكم روابط شرح لما جاء في الدرس :

https://youtu.be/Izwd_n-Ufqo?si=JSl6Kp8J2fJ1v7II
https://youtu.be/CCMKMBGUxkc?si=elSnhgk_sHIeWaaO
https://youtu.be/61i7VvPLVns?si=Vma1NjUPXYUZk5TJ
https://youtu.be/BNXasw_j4sY?si=0vVWlXimXi5FUeBC
https://youtu.be/7o58LMti2po?si=LPOBKioVJ-Nj3KLe
https://youtu.be/VQHLn1wuDBw?si=X-rt5rT_etDueukn
https://youtu.be/zFVdMyr6CIo?si=IpeoxqjNyZgkK9OF

في النهاية , هناك موضوع لم نتحدث عنه و ذلك لانه ليس بالشيء الصعب أو المهم , و لكن سنترك لكم رابطاً يشرح الموضوع و هو للاطلاع فقط :

https://youtu.be/oNp5wwu9S7c?si=VemdJXJ5vZDGSyMQ

PYTHON COURSE

Python course , from scratch to profissionalism ...

Made by : Taha Khaldoun Amro