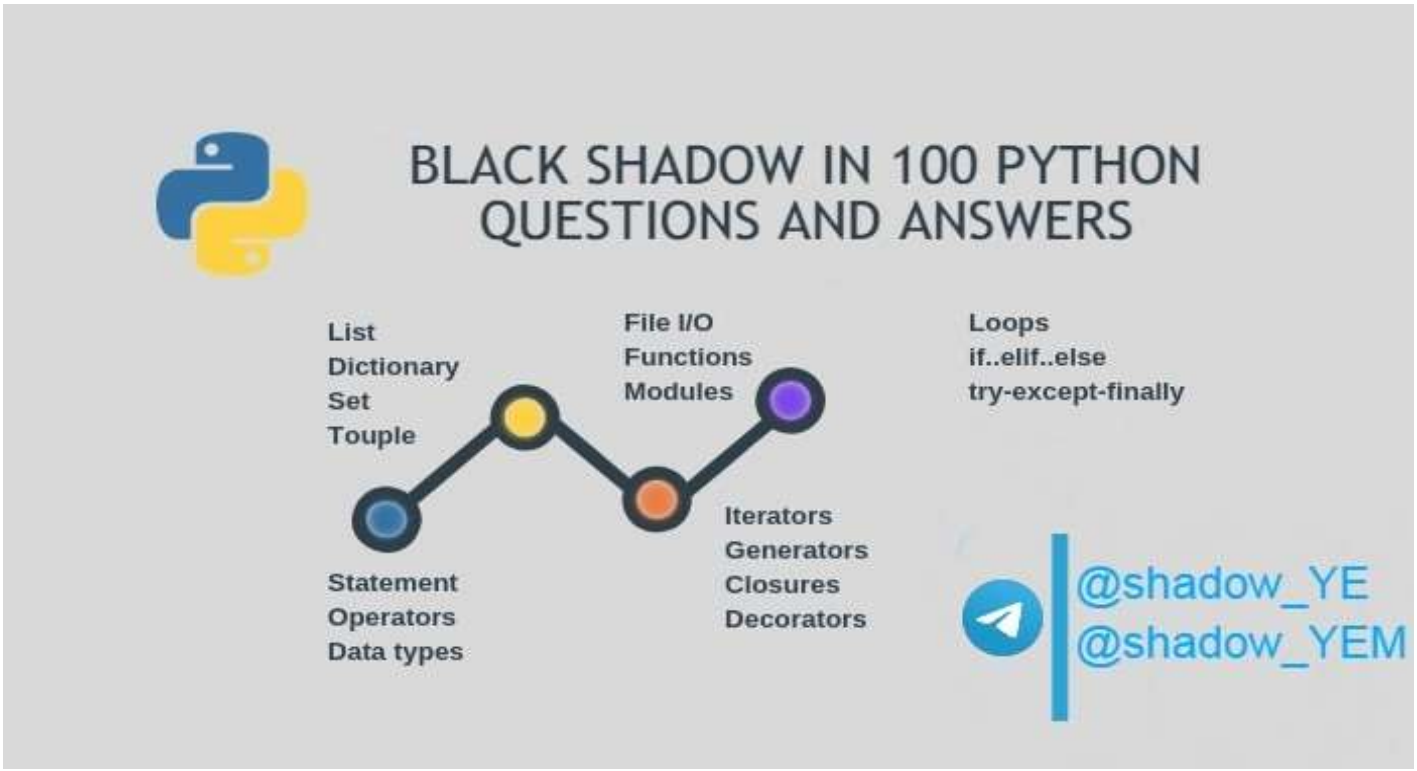
100 questions and answers in Python

SHADOW


BLACK SHADOW IN 100 PYTHON QUESTIONS AND ANSWERS
List
Dictionary
Set
Touple
File I/O
Functions
Modules
Loops
if..elif..else
try-except-finally
Iterators
Generators
Closures
Decorators
Statement
Operators
Data types
@shadow_YE
@shadow_YEM

مقدمة عن بايثون

1. لغة بايثون :

هي لغة برمجة عالية المستوى تتميز ببساطة كتابتها وقراءتها، سهلة التعلم، تستخدم أسلوب البرمجة الكائنية وهي مفتوحة المصدر وقابلة للامتداد.

تعتبر لغة بايثون لغة مفسرة متعددة الاستخدامات وتستخدم بشكل واسع في العديد من المجالات كبناء البرامج المستقلة باستخدام الواجهات الرسومية المعروفة وفي عمل تطبيقات الويب بالإضافة إلى استخدامها كلغة برمجة نصية للتحكم في أداء بعض من أشهر البرامج المعروفة أو في بناء برامج ملحقة لها.

وبشكل عام يمكن استخدام بايثون لبرمجة البرامج البسيطة للمبتدئين، ولإنجاز المشاريع الضخمة كأي لغة برمجية أخرى في نفس الوقت غالباً ما يُنصح المبتدؤون في ميدان البرمجة بتعلم هذه اللغة لأنها من بين أسرع اللغات البرمجية تعلماً

متى نشأت بايثون :

نشأت بايثون في معهد الرياضيات و المعلوماتية الهولندي (C WI) بأمستردام على يد جايدو ڤان روسم في أواخر الثمانينات من القرن المنصرم، وكان أول إعلان عنها في عام 1991. كتبت نواة اللغة بلغة سي .أطلق ڤان روسم الاسم "بايثون" على لغته تعبيرًا عن إعجابه بفِرقَة مسرحية هزلية شهيرة من بريطانيا، كانت تطلق على نفسها الاسم مونتي بايثون.

تتميز بايثون بمجتمعها النشط، كما أن لها الكثير من المكتبات البرمجية ذات الأغراض الخاصة والتي برمجها أشخاص من مجتمع هذه اللغة، مثلاً مكتبة باي جايم التي توفر مجموعة من الوظائف من أجل برمجة الألعاب .ويمكن لبايثون التعامل مع العديد من أنواع قواعد البيانات مثل ماي إس كيو إل وغيره. تدعم بايثون بارادايمات برمجية متعددة مثل التوجيه الكائني، البرمجية جانبية التوجيه و البرمجة الوظيفية .كما تقدم التنويع الديناميكي .تستخدم بايثون عادةً مثل العديد من لغات البرمجة الديناميكية كلغة برمجة نصية. بايثون لديها نموذج مفتوح للتطوير، القائم على مجتمع بايثون البرمجي والمدعوم أيضاً من مؤسسة برمجيات بايثون .والتي تحافظ على تعريف اللغة في التنفيذ المرجعي لسي بايثون.

2. ماهي مميزات لغة بايثون :

- تعد بايثون لغة برمجية سهلة نسبيا بالمقارنة مع جافا و C++ اضافة الى انها لغة برمجة متعددة الأنماط الفكرية(برمجة متعددة البرادايم)
- تدعم البرمجة كائنية التوجه والبرمجة المهيكلة بشكل كامل، كما تدعم بايثون البرمجة الوظيفية والبرمجة جانبية المنحى (بما في ذلك عن طريق البرمجة الوصفية والكائنات الوصفية ((الطرق السحرية))
- يمكنها أيضًا دعم العديد من الأنماط الفكرية الأخرى عن طريق الامتدادات، بما في ذلك التصميم بالعقود والبرمجة المنطقية ما في ذلك التصميم بالعقود والبرمجة المنطقية.

س 1: ما هي بايثون ، وما هي فوائد استخدامه ، وماذا تفهم من **PEP8**؟

الجواب :

بايثون النصي ، لا يلزم تجميعه قبل التنفيذ. بايثون هي واحدة من أنجح اللغات المفسرة. عند كتابة نص اللغات المفسرة الأخرى هي**Java script** و **PHP**عدد قليل من

فوائد برمجة بايثون:

- بايثون لغة كتابة ديناميكية. هذا يعني أنك لست بحاجة إلى ذكر نوع البيانات من المتغيرات أثناء إعلانها. يسمح بتعيين متغيرات مثل :

var1 = 101 and var2 = "You are an engineer."

بدون اي خطأ

- يدعم بايثون البرمجة الشيئية حيث يمكنك تحديد الفئات مع التركيب والميراث. لا يستخدم محددات الوصول مثل العامة أو الخاصة.

- الوظائف في بايثون تشبه كائنات الدرجة الأولى. يقترح عليك تعيينها للمتغيرات ، والعودة من الطرق الأخرى وتمريرها كوسيطة.

- يعد التطوير باستخدام بايثون سريعًا ولكن تشغيله غالبًا ما يكون أبطأ من اللغات المترجمة. لحسن الحظ ، تمكن بايثون من تضمين ملحقات اللغة "C" حتى تتمكن من تحسين البرامج النصية الخاصة بك.

- تمتلك بايثون العديد من الاستخدامات مثل التطبيقات المستندة إلى الويب وأتمتة الاختبار ونمذجة البيانات وتحليلات البيانات الضخمة والمزيد. بدلاً من ذلك ، يمكنك استخدامه كطبقة "غراء" للعمل مع لغات أخرى.

ماهو PEP 8 ؟

PEP 8 هو أحدث معيار ترميز بايثون ، مجموعة من توصيات الترميز. إنه يرشد لتقديم كود بايثون أكثر قابلية للقراءة.

س 2: ما هو إخراج جزء رمز بايثون التالي؟ برر جوابك.

```
def extendList(val, list=[]):
    list.append(val)
    return list
list1 = extendList(10)
list2 = extendList(123,[])
list3 = extendList('a')
print ("list1 = %s" % list1)
print ("list2 = %s" % list2)
print ("list3 = %s" % list3)
```

نتيجة مقتطف رمز بايثون أعلاه هو:

```
list1 = [10, 'a']
list2 = [123]
list3 = [10, 'a']
```

قد تتوقع خطأً أن تكون القائمة 1 مساوية لـ [10] وقائمة 3 لمطابقة ["a"] ، معتقدًا أن وسيطة القائمة ستتم تهيئتها لقيمتها الافتراضية [ ] في كل مرة يتم فيها استدعاء extList.

ومع ذلك ، فإن التدفق يشبه إنشاء قائمة جديدة مرة واحدة بعد تحديد الوظيفة.

ونفس الشيء يتم استخدامه عند استدعاء شخص لطريقة extList بدون وسيطة قائمة.

يعمل مثل هذا لأن حساب التعبيرات (في الوسيطات الافتراضية) يحدث في وقت تعريف الوظيفة ، وليس أثناء الاستدعاء.

ومن ثم تعمل القائمة 1 والقائمة 3 على نفس القائمة الافتراضية ، في حين تعمل القائمة 2 على كائن

منفصل أنشأته بنفسها (بتمرير قائمة فارغة كقيمة معلمة القائمة).

يمكن تغيير تعريف وظيفة extensionList بالطريقة التالية.

```
def extendList(val, list=None):
  if list is None:
    list = []
  list.append(val)
  return list
```

مع هذا التنفيذ المنقح ، سيكون الناتج:

```
list1 = [10]
list2 = [123]
list3 = ['a']
```

س 3: ما هي العبارة التي يمكن استخدامها في بايثون إذا لم يتطلب البرنامج أي إجراء ولكنه يتطلبها بشكل نحوي؟

عبارة المرور هي عملية فارغة. لا يحدث شيء عند تنفيذه. يجب عليك استخدام الكلمة الأساسية "pass" بأحرف صغيرة. إذا كتبت "Pass,"فستواجه خطأً مثل "NameError: name Pass is not defined." عبارات بايثون حساسة لحالة الأحرف

```
letter = "hai sethuraman"
for i in letter:
    if i == "a":
        pass
        print("pass statement is execute ..............")
    else:
        print(i)
```

س 4: ما هي عملية الحصول على الدليل الرئيسي باستخدام '~' في بايثون ؟

تحتاج إلى استيراد وحدة نظام التشغيل ، وبعد ذلك فقط سطر واحد يقوم بالباقي.

```
import os
print (os.path.expanduser('~'))
```

الناتج :

```
/home/runner
```

س 5: ما هي الأنواع المضمنة المتوفرة في بايثون ؟

فيما يلي قائمة بالأنواع المدمجة الأكثر استخدامًا والتي تدعمها بايثون:

- أنواع بيانات مدمجة غير قابلة للتغيير في بايثون
  - Numbers
  - Strings
  - Tuples
- أنواع بيانات مدمجة قابلة للتغيير في بايثون
  - List
  - Dictionaries
  - Sets

س 6: كيفية العثور على أخطاء أو إجراء تحليل ثابت في تطبيق بايثون ؟

- يمكنك استخدام PyChecker ، وهو محلل ثابت. يحدد الأخطاء في مشروع بايثون ويكشف أيضًا عن الأخطاء المتعلقة بالأسلوب والتعقيد.
- أداة أخرى هي بايثون ، والتي تتحقق مما إذا كانت وحدة Python تفي بمعيار الترميز.

س 7: متى يستخدم ديكور بايثون؟

Python decorator هو تغيير نسبي تقوم به في بناء جملة بايثون لضبط الوظائف بسرعة.

س 8: ما هو الفرق الرئيسي بين القائمة والمجموعة؟ List vs Tuple

والفرق الرئيسي بين القائمة والمجموعة هو أن الأول قابل للتغيير بينما لا يتغير.

يُسمح بتجزئة tuple ، على سبيل المثال ، استخدامه كمفتاح للقواميس.

س 9: كيف تتعامل بايثون مع إدارة الذاكرة؟

- تستخدم بايثون أكوامًا خاصة للحفاظ على ذاكرتها. لذا يحمل الكومة جميع كائنات بايثون وهياكل البيانات. هذه المنطقة متاحة فقط لمترجم بايثون. المبرمجون لا يستخدمونه.
- ومدير ذاكرة بايثون هو الذي يعالج الكومة الخاصة. يقوم بالتخصيص المطلوب للذاكرة لكائنات بايثون.
- تستخدم Python جامع قمامة مدمجًا ، والذي ينقذ كل الذاكرة غير المستخدمة ويفرغها إلى مساحة كومة الذاكرة المؤقتة.

س 10: ما هي الاختلافات الرئيسية بين lambda and def ؟ Lambda vs def

- يمكن لـ Def حمل العديد من التعبيرات في حين أن lambda هي وظيفة أحادية التعبير.
- يقوم Def بإنشاء وظيفة وتعيين اسم لاستدعائها لاحقًا. تشكل لامدا كائن دالة وتعيدها.
- يمكن أن يكون لـ Def بيان إرجاع. لا يمكن أن يكون لامدا عبارات الإرجاع.
- يدعم Lambda التعود داخل قائمة وقاموس.

س 11: اكتب تعبير reg الذي يؤكد معرف البريد الإلكتروني باستخدام وحدة تعبير python reg "re"؟

يحتوي بايثون على وحدة تعبير عادية ".re"

تحقق من تعبير "re" الذي يمكنه التحقق من معرف البريد الإلكتروني للنطاق الفرعي .com و. co.in.

```
import re

print(re.search(r"[0-9a-zA-Z.]+@[a-zA-
Z]+\.(com|co\.in)$","micheal.pages@mp.com"))
```

س 12: ما رأيك في إخراج جزء التعليمات البرمجية التالي؟ هل يوجد خطأ في الكود؟

```
list = ['a', 'b', 'c', 'd', 'e']

print (list[10:])
```

نتيجة أسطر التعليمات البرمجية أعلاه []. لن يكون هناك أي خطأ مثل خطأ الفهرس.

يجب أن تعلم أن محاولة جلب عضو من القائمة باستخدام فهرس يتجاوز عدد الأعضاء (على سبيل المثال ، محاولة الوصول إلى القائمة [10] كما هو مذكور في السؤال) سيؤدي إلى خطأ في الفهرس. بالمناسبة ، استرجاع شريحة فقط في فهرس البداية الذي يتجاوز رقم. من العناصر الموجودة في القائمة لن ينتج عنها خطأ في الفهرس. ستقوم فقط بإرجاع قائمة فارغة.

س 13: هل هناك تبديل أو بيان حالة في بايثون ؟ إذا لم يكن كذلك فما هو السبب؟

الإجابة :

لا ، ليس لدى Python بيان Switch ، ولكن يمكنك كتابة دالة Switch ثم استخدامها.

س 14: ما هي الوظيفة المضمنة التي تستخدمها بايثون لتكرارها عبر تسلسل رقمي؟

يقوم () Range بإنشاء قائمة بأرقام يتم استخدامها لتكرار الحلقات.

```
for i in range(5):
    print(i)
```

ترافق الدالة ()range مجموعتين من المعلمات.

- range(stop)
  - stop : إنه لا. من الأعداد الصحيحة المراد إنشاؤها ويبدأ من الصفر. على سبيل المثال

    range(3) == [0, 1, 2]

- range([start], stop[, step])
  - Start: التسلسل لا. إنها البداية.
  - Stop: يحدد الحد الأعلى للتسلسل.
  - Step: هي العامل المتزايد لتوليد التسلسل.
- نقاط للملاحظة:
  - يسمح فقط الوسيطات الصحيحة.
  - يمكن أن تكون المعلمات موجبة أو سالبة.

- ال()range تبدأ الدالة في Python من مؤشر zeroth.

س 15: ما هي العبارات الاختيارية الممكنة داخل كتلة باستثناء في بايثون؟

هناك بندان اختياريان يمكنك استخدامهما في المحاولة باستثناء الكتلة.

- The "**else**" clause
  - من المفيد إذا كنت ترغب في تشغيل جزء من التعليمات البرمجية عندما لا يقوم قالب المحاولة بإنشاء استثناء.
- The "**finally**" clause
  - يفيد عندما تريد تنفيذ بعض الخطوات التي تعمل ، بغض النظر عما إذا كان هناك استثناء أم لا.

س 16: ما هو السلسلة في بايثون؟

السلسلة في بايثون هي سلسلة من الأحرف الأبجدية الرقمية. إنها أشياء ثابتة. هذا يعني أنهم لا يسمحون بالتعديل بمجرد الحصول على قيمة. يوفر بايثون عدة طرق ،

مثل ()join أو ()replace أو ()split لتغيير السلاسل. ولكن لا شيء من هذا يغير الكائن الأصلي.

س 17: ما هو التقطيع في بايثون؟

التقطيع هو عملية سلسلة لاستخراج جزء من السلسلة أو جزء من القائمة. . في Python ، تبدأ سلسلة (say text) في الفهرس 0 ، ويخزن الحرف n في نص الموضع [n-1]. يمكن لـ Python أيضًا إجراء فهرسة عكسية ، أي في الاتجاه الخلفي ، بمساعدة الأرقام السالبة. في Python ، تعد ()slice (start, stop, step)

تسمح طريقة ()slice بثلاث معلمات.

1. البدء – رقم البداية لبدء التقطيع.

2. توقف – الرقم الذي يشير إلى نهاية التقطيع.

3. الخطوة – القيمة لزيادة بعد كل فهرس (1 = default).

س 18: ما هو %s في بايثون؟

يدعم Python تنسيق أي قيمة في سلسلة. قد تحتوي على تعبيرات معقدة للغاية.

تتمثل إحدى الاستخدامات الشائعة في دفع القيم إلى سلسلة باستخدام محدد تنسيق٪ s. تحتوي عملية التنسيق في بايثون على بنية مماثلة كما في لغة C لما تحتويه الدالة ()printf

س 19: هل السلسلة ثابتة أو قابلة للتغيير في بايثون؟

سلاسل بايثون ثابتة بالفعل . دعنا نأخذ مثال. لدينا متغير "str" يحمل قيمة سلسلة. لا يمكننا تحوير الحاوية ، أي السلسلة ، ولكن يمكننا تعديل ما تحتوي عليه وهذا يعني قيمة المتغير.

س 20: ما هو الفهرس في بايثون؟

الفهرس هو نوع بيانات عدد صحيح يشير إلى موضع داخل قائمة مرتبة أو سلسلة.

السلاسل في بايثون هي أيضًا قوائم بالشخصيات. يمكننا الوصول إليهم باستخدام الفهرس الذي يبدأ من الصفر ويصل إلى الطول ناقص واحد.

على سبيل المثال ، في السلسلة "Program" ، تحدث الفهرسة على النحو التالي:

```
Program 0 1 2 3 4 5
```

س 21: ما هو Docstring في بايثون؟

docstring هو نص فريد يحدث ليكون العبارة الأولى في بنيات Python التالية:

تعريف الوحدة أو الوظيفة أو الفئة أو الطريقة.

تتم إضافة docstring إلى سمة __doc__ لكائن السلسلة.

الآن ، اقرأ بعض أسئلة مقابلة بايثون حول الوظائف.

س 22: ما هي الوظيفة في برمجة بايثون ؟

الوظيفة هي كائن يمثل كتلة من التعليمات البرمجية وهو كيان قابل لإعادة الاستخدام. يجلب الوحدات النمطية لبرنامج ودرجة أعلى من قابلية إعادة استخدام التعليمات البرمجية.

لقد أعطتنا بايثون العديد من الوظائف المضمنة مثل ()print وتوفر القدرة على إنشاء وظائف معرفة من قبل المستخدم.

س 23: كم عدد أنواع الوظائف الأساسية المتوفرة في بايثون ؟

تعطينا Python نوعين أساسيين من الوظائف.

1- Built-in, and
2- User-defined.

تصادف أن تكون الوظائف المضمنة جزءًا من لغة بايثون. بعض هذه

()print و ()dir و ()len و ()abs إلخ.

س 24: كيف نكتب دالة في بايثون؟

يمكننا إنشاء دالة بايثون بالطريقة التالية.

الخطوة 1: لبدء الوظيفة ، ابدأ الكتابة باستخدام الكلمة المفتاحية def ثم اذكر اسم الوظيفة.

الخطوة 2: يمكننا الآن تمرير الحجج وإرفاقها باستخدام الأقواس. تشير النقطتين ، في النهاية ، إلى نهاية رأس الوظيفة.

الخطوة 3: بعد الضغط على مفتاح الإدخال ، يمكننا إضافة عبارات بايثون المطلوبة للتنفيذ.

س 25: ما هو استدعاء دالة أو كائن قابل للاستدعاء في بايثون ؟

يتم التعامل مع دالة في بايثون ككائن قابل للاستدعاء. يمكن أن يسمح ببعض الحجج ويعيد أيضًا قيمة أو قيمًا متعددة في شكل مجموعة. بصرف النظر عن الوظيفة ، تحتوي بايثون على تراكيب أخرى ، مثل الفئات أو المثيلات الطبقية التي تناسب نفس الفئة.

س 26: ما هي الكلمة الرئيسية المستخدمة في بايثون ؟

الغرض من الوظيفة هو استقبال المدخلات وإرجاع بعض المخرجات.

العائد عبارة عن بيان بايثون يمكننا استخدامه في دالة لإرسال قيمة إلى المتصل.

س 27: ما هو "Call by Value" في بايثون ؟

في الاستدعاء بالقيمة ، الوسيطة سواء كان التعبير أو القيمة منضمًا إلى المتغير المعني في الوظيفة. ستتعامل بايثون مع هذا المتغير كمحلي في نطاق مستوى الوظيفة. ستظل أي تغييرات يتم إجراؤها على هذا المتغير محلية ولن تنعكس خارج الوظيفة.

س 28: ما هو "Call by Reference" في بايثون؟

نحن نستخدم كلاً من "استدعاء بالإشارة" و "المرور بالإشارة" بالتبادل. عندما نقوم بتمرير وسيطة حسب المرجع ، تكون متاحة كمرجع ضمني للدالة ، بدلاً من نسخة بسيطة. في هذه الحالة ، سيكون أي تعديل على الوسيطة مرئيًا أيضًا للمتصل.

يتمتع هذا المخطط أيضًا بميزة توفير المزيد من الوقت والفعالية في المساحة لأنه يترك الحاجة إلى إنشاء نسخ محلية.

على العكس من ذلك ، قد يكون العيب أن المتغير يمكن أن يتغير عن طريق الخطأ أثناء استدعاء دالة. وبالتالي ، يحتاج المبرمجون إلى التعامل مع التعليمات البرمجية لتجنب مثل هذه الشكوك.

س 29: ما هي القيمة المرجعة للدالة ()trunc ؟

تقوم دالة ()Python trunc بتنفيذ عملية حسابية لإزالة القيم العشرية من تعبير معين وتوفر قيمة عددية كمخرجاتها.

س 30: هل من الضروري أن تقوم دالة بايثون بإرجاع قيمة؟

ليس من الضروري على الإطلاق أن تُرجع الدالة أي قيمة. ومع ذلك ، إذا لزم الأمر ، يمكننا استخدام بلا كقيمة إرجاع.

س 31: ماذا يفعل الاستمرار في بايثون؟

المتابعة عبارة عن عبارة قفزة في بايثون والتي تنقل عنصر التحكم لتنفيذ التكرار التالي في حلقة تاركة جميع التعليمات المتبقية في الكتلة بدون تنفيذ. بيان المتابعة قابل للتطبيق لكل من حلقات "while" و "for".

س 32: ما هو الغرض من دالة ()id في بايثون ؟

()id هي إحدى الوظائف المضمنة في بايثون.

```
Signature: id(object)
```

يقبل معلمة واحدة ويعيد معرّفًا فريدًا مرتبطًا بكائن الإدخال.

س33: What does the *args do in Python؟

نستخدم *args كمعلمة في رأس الدالة. يمنحنا القدرة على تمرير عدد N (متغير) من الحجج.

يرجى ملاحظة أن هذا النوع من بناء الجملة الوسيطة لا يسمح بتمرير وسيطة مسماة إلى الوظيفة.

مثال على استخدام *args:

```
# Python code to demonstrate
# *args for dynamic arguments
def fn(*argList):
    for argx in argList:
        print (argx)

fn('I', 'am', 'Learning', 'Python')
```

الإخراج:

```
I
am
Learning
Python
```

س 34: ما الذي يفعله **kwargs في بايثون ؟

. يمكننا أيضًا استخدام بناء جملة في **kwargs إعلان دالة Python. يتيح لنا تمرير عدد (variable) N من الوسيطات التي يمكن تسميتها أو الكلمات الرئيسية.

مثال على استخدام **kwargs:

```
# Python code to demonstrate

# **kwargs for dynamic + named arguments

def fn(**kwargs):

    for emp, age in kwargs.items():

        print ("%s's age is %s." %(emp, age))


fn(Gilan=17, nghm=18, shadow=20)
```

الإخراج:

```
Gilan's age is 17.

nghm's age is 18.

shadow's age is 20 .
```

س 35: هل لدى بايثون طريقة Main() ؟

main() هي وظيفة نقطة الدخول التي يتم استدعاؤها أولاً في معظم لغات البرمجة.

نظرًا لأن بايثون قائم على المترجم ، فإنه ينفذ تسلسل التعليمات البرمجية واحدًا تلو الآخر.

لدى بايثون أيضًا طريقة Main(). ولكن يتم تنفيذه كلما قمنا بتشغيل نص بايثون النصي الخاص بك إما عن طريق النقر عليه مباشرة أو تشغيله من سطر الأوامر.

يمكننا أيضًا تجاوز الإعداد الافتراضي لـ بايثون main() تعمل باستخدام بايثون إذا البيان. يرجى الاطلاع على الرمز أدناه.

```
print("Welcome")
print("__name__ contains: ", __name__)
def main():
    print("Testing the main function")
if __name__ == '__main__':
    main()
```

الإخراج:

```
Welcome
__name__ contains:  __main__
Testing the main function
```

س 36 : ماذا يفعل __ Name __ في بايثون ؟

الـ __ Name __ متغير فريد. نظرًا لأن بايثون لا تعرض وظيفة main() لذلك عندما يقوم مترجمة بتشغيل البرنامج النصي في الدالة () ، لذا عندما يقوم مترجمة بتشغيل البرنامج النصي ، تقوم أولاً بتنفيذ التعليمات البرمجية في المستوى 0 المسافة الفارغة. لمعرفة ما إذا تم استدعاء main() الرئيسي ، يمكننا استخدام متغير __name__ في عبارة if مقارنة بالقيمة "__main__".

س 37: ما هو الغرض من "end" في بايثون؟

تقوم وظيفة print() في بايثون دائمًا بطباعة سطر جديد في النهاية. تقبل الدالة print() معلمة اختيارية تُعرف باسم "end". قيمته هي "\n" بشكل افتراضي. يمكننا تغيير الحرف النهائي في عبارة طباعة بالقيمة التي تختارها باستخدام هذه المعلمة.

```
# Example: Print a  instead of the new line in the
end.

print("Let's learn" , end = ' ')

print("Python")


# Printing a dot in the end.

print("Browse through Google" , end = '.')

print("com", end = ' ')
```

الإخراج:

```
Let's learn Python

Browse through  google.com
```

س 38: متى يجب استخدام "break" في بايثون ؟

تقدم بايثون بيان فاصل للخروج من حلقة. عندما يضرب الكسر في الكود ، يخرج عنصر التحكم في البرنامج على الفور من نص الحلقة. تؤدي العبارة الفاصلة في حلقة متداخلة إلى خروج عنصر التحكم من الكتلة التكرارية الداخلية.

س 39: ما الفرق بين pass و continue في بايثون؟

continue يجعل البيان الحلقة للاستئناف من التكرار التالي.

على العكس من ذلك ، يرشد بيان الـ pass بعدم القيام بأي شيء ، ويتم تنفيذ ما تبقى من التعليمات البرمجية كالمعتاد.

تعريف Continue و pass

: Continue

ترجع عبارة continue في بايثون عنصر التحكم إلى بداية حلقة while يرفض بيان المتابعة جميع العبارات المتبقية في التكرار الحالي للحلقة وينقل عنصر التحكم إلى أعلى الحلقة. يمكن استخدام عبارة المتابعة في كل من الوقت والتكرار.

: Pass

يتم استخدام عبارة pass كعنصر نائب للرمز المستقبلي. عند تنفيذ عبارة pass، لا يحدث شيء ، لكنك تتجنب الحصول على خطأ عندما لا يُسمح برمز فارغ. لا يُسمح بالشفرة الفارغة في الحلقات أو تعريفات الدوال أو تعريفات الصنف أو في عبارات if.

س 40: ماذا تفعل دالة len() في بايثون ؟

في بايثون ، تعد len() دالة سلسلة أساسية. يحدد طول سلسلة الإدخال.

```
>>> some_string = 'BLACK SHADOW'
>>> len(some_string)
12
```

س 41: ماذا تفعل وظيفة ()chr في بايثون؟

تمت إعادة إضافة وظيفة ()chr في Python 3.2. في الإصدار 3.0 ، تمت إزالته.

تقوم بإرجاع السلسلة التي تشير إلى حرف تكون نقطة رمز Unicode الخاصة به عددًا صحيحًا.

فمثلا، يُرجع (122) chr السلسلة "z" بينما يُرجع (1212) chr السلسلة "e".

```
>>> ord("z")
122
```

س 43: ما هو ()Rstrip في بايثون ؟

يوفر بايثون أسلوب ()rstrip الذي يكرر السلسلة لكنه يترك أحرف المسافة البيضاء من النهاية.

يهرب ()rstrip من الأحرف من الطرف الأيمن بناءً على قيمة الوسيطة ، أي سلسلة تشير إلى مجموعة الأحرف المطلوب استبعادها.

توقيع ()rstrip هو:

```
str.rstrip([char sequence/pre>
```

```
#Example
test_str = 'Programming    '
# The trailing whitespaces are excluded
print(test_str.rstrip())
```

س 44: ما هي المسافة البيضاء في بايثون؟

تمثل المسافة البيضاء الأحرف التي نستخدمها للمسافة والفصل.

لديهم تمثيل "empty". في بايثون ، يمكن أن تكون علامة تبويب أو مسافة.

س 45: ما هو isalpha() في بايثون؟

توفر بايثون هذه الوظيفة isalpha() المضمنة لغرض معالجة السلسلة.

تقوم بإرجاع True إذا كانت جميع الأحرف في السلسلة من نوع الأبجدية ، وإلا فإنها تُرجع False.

س 46: كيف تستخدم الدالة split() في بايثون ؟

توفر بايثون هذه الوظيفة isalpha () المضمنة لغرض معالجة السلسلة. تقوم بإرجاع True إذا كانت جميع الأحرف في السلسلة من نوع الأبجدية ، وإلا فإنها تُرجع False.

س 46: كيف تستخدم الدالة split() في بايثون ؟

تعمل وظيفة بايثون split() على الأوتار لقطع قطعة كبيرة إلى قطع أصغر ، أو سلاسل فرعية. يمكننا تحديد فاصل لبدء التقسيم ، أو أنه يستخدم المساحة كواحد افتراضيًا.

```
#Example

str = 'pdf csv json'

print(str.split(" "))

print(str.split())
```

الإخراج:

```
['pdf', 'csv', 'json']

['pdf', 'csv', 'json']
```

س 47: ماذا تفعل طريقة ()join في بايثون؟

توفر بايثون طريقة ()join التي تعمل على السلاسل والقوائم والصفوف. فهو يجمعهم ويعيد قيمة موحدة.

س 48: ماذا تفعل طريقة ()Title في بايثون؟

توفر بايثون طريقة ()title لتحويل الحرف الأول في كل كلمة إلى تنسيق كبير بينما يتحول الباقي إلى أحرف صغيرة.

```
#Example

str = 'lEaRn pYtHoN'

print(str.title())
```

الإخراج:

```
Learn Python
```

س 49: ما الذي يجعل CPython يختلف عن بايثون ؟

تم تطوير جوهر CPython في C. تمثل البادئة "C" هذه الحقيقة. يقوم بتشغيل حلقة مترجم تستخدم لترجمة كود Python-ish إلى لغة C.

س 50: أي حزمة هي أسرع شكل لبيثون؟

يوفر PyPy أقصى توافق أثناء استخدام تطبيق CPython لتحسين أدائه.

أكدت الاختبارات أن PyPy أسرع بخمس مرات من CPython. وهو يدعم حاليًا Python 2.7.

س 51: ما هو GIL في لغة بايثون ؟

يدعم Python GIL (قفل المترجم العالمي) وهو عبارة عن كائن مزامنة يُستخدم لتأمين الوصول إلى كائنات Python ، ومزامنة سلاسل محادثات متعددة من تشغيل رموز بايثون الفرعية في نفس الوقت.

س 52: كيف يكون خيط بايثون آمنًا؟

يضمن بايثون الوصول الآمن إلى سلاسل المحادثات. يستخدم كائن مزامنة GIL لضبط المزامنة. إذا فقد الخيط قفل GIL في أي وقت ، فيجب عليك جعل الخيط آمنًا.

على سبيل المثال ، يتم تنفيذ العديد من عمليات بايثون على أنها ذرية مثل طريقة استدعاء ()sort في القائمة.

س 53: كيف تدير بايثون الذاكرة؟

تقوم بايثون بتنفيذ مدير كومة داخليًا يحتفظ بكافة كائناته وهياكل البيانات الخاصة به.

يقوم مدير الكومة هذا بتخصيص / إلغاء تخصيص مساحة كومة الذاكرة المؤقتة للكائنات.

س 54: ما هو tuple في بايثون؟

tuple هو بنية بيانات من نوع المجموعة في بايثون وهي غير قابلة للتغيير.

إنها تشبه التسلسلات ، تمامًا مثل القوائم. ومع ذلك ، هناك بعض الاختلافات بين مجموعة وقائمة ؛ السابق لا يسمح بالتعديلات بينما القائمة.

أيضًا ، تستخدم tuples بين قوسين للتضمين ، ولكن القوائم تحتوي على أقواس مربعة في تركيبها.

س 55: ما هو Dictionary في برمجة بايثون ؟

Dictionary :هو بنية بيانات تُعرف باسم مصفوفة ترابطية في بايثون تقوم بتخزين مجموعة من الكائنات.

Collection : هي مجموعة من المفاتيح لها قيمة مرتبطة واحدة. يمكننا تسميتها علامة تجزئة أو خريطة أو خريطة تجزئة كما يتم استدعاؤها بلغات برمجة أخرى.

س 56: ما هو مجموعة object في بايثون؟

المجموعات هي كائنات مجموعة غير مرتبة في بايثون. يخزنون أشياء فريدة وثابتة. بايثون له تطبيق مشتق من الرياضيات.

س 57: ما هو استخدام القاموس في بايثون؟

يحتوي القاموس على مجموعة من الكائنات (المفاتيح) يتم تعيينها لمجموعة أخرى من الكائنات (القيم). يمثل قاموس بايثون تعيين مفاتيح فريدة للقيم.

إنها قابلة للتغيير وبالتالي لن تتغير. يمكن أن تكون القيم المرتبطة بالمفاتيح من أي نوع من أنواع بايثون.

س 58: هل قائمة بايثون قائمة مرتبطة؟

قائمة بايثون هي صفيف متغير الطول يختلف عن القوائم المرتبطة بنمط C.

داخليًا ، يحتوي على مصفوفة متجاورة للإشارة إلى كائنات أخرى ويخزن مؤشرًا لمتغير المصفوفة وطوله في بنية رأس القائمة.

فيما يلي بعض أسئلة في بايثون حول classes و objects.

س 59: ما هو class في بايثون؟

يدعم بايثون البرمجة الموجهة للكائنات ويوفر تقريبًا جميع ميزات OOP لاستخدامها في البرامج.

فئة بايثون هي مخطط لإنشاء الكائنات. يحدد متغيرات الأعضاء ويحصل على سلوكهم المرتبط بها.

يمكننا أن نجعلها باستخدام الكلمة "class". يتم إنشاء كائن من المنشئ. يمثل هذا الكائن مثيل class.

في بايثون ، نقوم بإنشاء فئات ومثيلات بالطريقة التالية.

```
>>>class Human:  # Create the class
...     pass
>>>man = Human()  # Create the instance
>>>print(man)
<__main__.Human object at 0x02E822C8>
```

س 60: ما هي الصفات والأساليب في class في بايثون؟

الـ class عديمة الفائدة إذا لم تحدد أي وظيفة. يمكننا القيام بذلك عن طريق إضافة السمات. تعمل كحاويات للبيانات والوظائف. يمكننا إضافة سمة تحدد مباشرة داخل نص الفئة.

```
>>> class Human:

...     profession = "programmer" # specify the attribute 'profession'
of the class

>>> man = Human()

>>> print(man.profession)

programmer
```

بعد إضافة السمات ، يمكننا الاستمرار في تحديد الوظائف. بشكل عام ، نسميها الأساليب. في توقيع الطريقة ، يتعين علينا دائمًا تقديم الوسيطة الأولى بكلمة رئيسية ذاتية.

```
>>> class Human:

    profession = "programmer"

    def set_profession(self, new_profession):

        self.profession = new_profession

>>> man = Human()

>>> man.set_profession("Manager")

>>> print(man.profession)

Manager
```

س 61: كيفية تعيين قيم لسمات class في وقت التشغيل؟

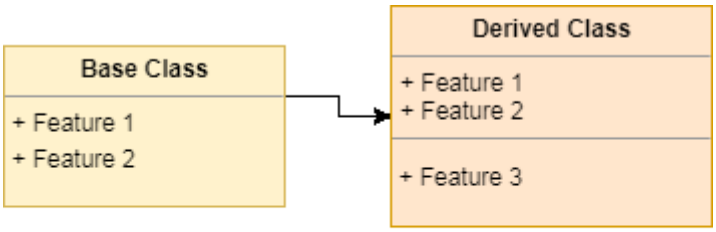
يمكننا تحديد قيم السمات في وقت التشغيل. نحن بحاجة إلى إضافة طريقة init و pass الإدخال إلى مُنشئ الكائن. انظر المثال التالي يوضح هذا.

```
>>> class Human:
    def __init__(self, profession):
        self.profession = profession
    def set_profession(self, new_profession):
        self.profession = new_profession


>>> man = Human("Manager")
>>> print(man.profession)
Manager
```

س 62: ما هو الميراث في برمجة بايثون؟

الوراثة ميزة قوية في البرمجة الشيئية. يشير إلى تعريف كلاس جديد مع تعديل بسيط أو معدوم للكلاس الموجودة يسمى الكلاس الجديد بـ derived (or child) class والذي يرث منه يسمى الـ **base (or parent) class**.



Inheritance In Python

نقوم بذلك عن عمد لاستخراج رمز مماثل في different classes.

تقع التعليمات البرمجية الشائعة على class الأساسية ، ويمكن لكائنات الكلاسات الفرعية الوصول إليها عبر الوراثة. تحقق من المثال أدناه.

```
class PC: # Base class
    processor = "Xeon" # Common attribute
    def set_processor(self, new_processor):
        processor = new_processor

class Desktop(PC): # Derived class
    os = "Mac OS High Sierra" # Personalized attribute
    ram = "32 GB"

class Laptop(PC): # Derived class
    os = "Windows 10 Pro 64" # Personalized attribute
    ram = "16 GB"

desk = Desktop()
print(desk.processor, desk.os, desk.ram)

lap = Laptop()
print(lap.processor, lap.os, lap.ram)
```

الإخراج :

```
Xeon Mac OS High Sierra 32 GB

Xeon Windows 10 Pro 64 16 GB
```

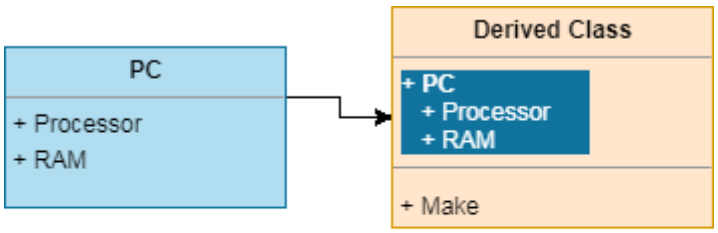
س 63: ما هو التكوين في بايثون؟

التكوين هو أيضًا نوع من الميراث في بايثون .ينوي أن يرث من الطبقة الأساسية ولكن بشكل مختلف قليلاً ،

بمعنى آخر.،

باستخدام متغير مثيل للقاعدة class تعمل كعضو في derived class

انظر الرسم البياني أدناه.



Composition In Python

لإثبات التكوين ، نحتاج إلى إنشاء كائنات أخرى في الفصل ثم الاستفادة من تلك الحالات

```
class PC: # Base class

    processor = "Xeon" # Common attribute

    def __init__(self, processor, ram):

        self.processor = processor

        self.ram = ram


    def set_processor(self, new_processor):

        processor = new_processor


    def get_PC(self):

        return "%s cpu & %s ram" % (self.processor, self.ram)


class Tablet():

    make = "Intel"

    def __init__(self, processor, ram, make):

        self.PC = PC(processor, ram) # Composition

        self.make = make


    def get_Tablet(self):

        return "Tablet with %s CPU & %s ram by %s" % (self.PC.processor,
self.PC.ram, self.make)


if __name__ == "__main__":

    tab = Tablet("i7", "16 GB", "Intel")

    print(tab.get_Tablet())
```

الإخراج :

```
Tablet with i7 CPU & 16 GB ram by Intel
```

س 64: ما هي الأخطاء والاستثناءات في برامج بايثون؟

الأخطاء هي مشكلات ترميز في البرنامج مما قد يؤدي إلى خروجها بشكل غير طبيعي

على العكس من ذلك ، تحدث الاستثناءات بسبب حدوث حدث خارجي يقطع التدفق الطبيعي للبرنامج

س 65: كيف تتعامل مع الاستثناءات باستخدام Try / Except / Finally في بايثون ؟

تضع بايثون تصميمات الـ "Try, Except, Finally" للتعامل مع الأخطاء والاستثناءات. نرفق الرمز غير الآمن الذي يحتوي على مسافة بادئة تحت ."try block" ويمكننا الاحتفاظ برمزنا الاحتياطي داخل الـ " except block".

يجب أن تأتي أي تعليمات معدة للتنفيذ أخيرًا ضمن "finally block"

```
    print("Executing code in the try block")
    print(exception)
except:
    print("Entering in the except block")
finally:
    print("Reached to the final block")
```

الإخراج:

```
Executing code in the try block

Entering in the except block

Reached to the final block
```

س 66: كيف ترفع الاستثناءات لشرط محدد مسبقًا في بايثون؟

يمكننا رفع استثناء بناء على بعض الشروط. على سبيل المثال ، إذا أردنا أن يقوم المستخدم بإدخال أرقام فردية فقط ، فسوف يثير استثناءً.

```
# Example - Raise an exception

while True:

    try:

        value = int(input("Enter an odd number- "))

        if value%2 == 0:

            raise ValueError("Exited due to invalid
input!!!")

        else:

            print("Value entered is : %s" % value)

    except ValueError as ex:

        print(ex)

        break
```

الاخراج :

```
Enter an odd number- 2

Exited due to invalid input!!!
```

```
Enter an odd number- 1

Value entered is : 1
```

س 67: ما هي بايثون التكرار؟

التكرارات في بايثون هي كائنات تشبه المصفوفة تسمح بالتحرك على العنصر التالي. نستخدمها في اجتياز حلقة ، على سبيل المثال ، في حلقة "for".

مكتبة بايثون لا يوجد. المكرر. على سبيل المثال ، تعد القائمة مكررًا أيضًا ويمكننا بدء حلقة للتكرار فوقها.

س 68: ما الفرق بين المكرر والمكرر؟

إن نوع المجموعة مثل القائمة ، list ، tuple، dictionary و set كلها كائنات قابلة للتكرار بينما هي أيضًا حاويات قابلة للتكرار والتي تعيد المكرر أثناء العبور.

فيما يلي بعض أسئلة مقابلة بايثون المتقدمة.

س 69: ما هي مولدات بايثون؟

Generator : هي نوع من الوظائف التي تتيح لنا تحديد وظيفة تعمل مثل المكرر وبالتالي يمكن استخدامها في حلقة "for".

في دالة generator ، تحل الكلمة الأساسية الخاصة بالإنتاج محل بيان الإرجاع.

مثال :

```
# Simple Python function

def fn():

    return "Simple Python function."


# Python Generator function

def generate():

    yield "Python Generator function."


print(next(generate()))
```

الإخراج :

```
Python Generator function.
```

س 70: ما هي عمليات الإغلاق في بايثون؟

إغلاق بايثون هي كائنات دالة يتم إرجاعها بواسطة دالة أخرى. نستخدمها للقضاء على تكرار التعليمات البرمجية. في المثال أدناه ، قمنا بكتابة إغلاق بسيط لمضاعفة الأرقام.

```
def multiply_number(num):

    def product(number):

        'product() here is a closure'

        return num * number

    return product


num_2 = multiply_number(2)

print(num_2(11))

print(num_2(24))


num_6 = multiply_number(6)

print(num_6(1))
```

الإخراج :

```
22

48

6
```

س 71: ما هي الديكورات في بايثون؟

القدرة على إضافة سلوك جديد إلى الكائنات المحددة ديناميكيًا. في المثال أدناه ، بايثون يمنحنا ديكور .كتبنا مثالًا بسيطًا لعرض رسالة قبل تنفيذ وظيفة ونشرها

```
def decorator_sample(func):
    def decorator_hook(*args, **kwargs):
        print("Before the function call")
        result = func(*args, **kwargs)
        print("After the function call")
        return result
    return decorator_hook

@decorator_sample
def product(x, y):
    "Function to multiply two numbers."
    return x * y

print(product(3, 3))
```

الإخراج :

```
Before the function call
After the function call
9
```

س 72: كيف تقوم بإنشاء قاموس في بايثون ؟

لنأخذ مثال بناء إحصاءات الموقع. لهذا ، نحتاج أولاً إلى تفكيك أزواج القيمة الرئيسية باستخدام النقطتين (":"). يجب أن تكون المفاتيح من النوع غير القابل للتغيير ، أي أننا سنستخدم أنواع البيانات التي لا تسمح بالتغييرات في وقت التشغيل. سنختار من بين int أو string أو tuple.

ومع ذلك ، يمكننا أن نأخذ قيمًا من أي نوع. للتمييز بين أزواج البيانات ، يمكننا استخدام فاصلة (",") والاحتفاظ بالأشياء بأكملها داخل أقواس متعرجة ({...}).

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000, "type":
"organic"}

>>> type(site_stats)

<class 'dict'>

>>> print(site_stats)



{'type': 'organic', 'site': 'tecbeamers.com', 'traffic': 10000}
```

س 73: كيف تقرأ من القاموس في بايثون؟

لجلب البيانات من القاموس ، يمكننا الوصول مباشرة باستخدام المفاتيح. يمكننا إرفاق "key" باستخدام الأقواس [...] بعد ذكر اسم المتغير المقابل للقاموس.

```
>>> site_stats = {'site': 'tecbeamers.com', 'traffic': 10000,
"type": "organic"}

>>> print(site_stats["traffic"])
```

يمكننا حتى استدعاء طريقة get لجلب القيم من الإملاء. كما سمح لنا بتعيين قيمة افتراضية. إذا كان المفتاح مفقودًا ، فسيحدث KeyError.

```
>>> site_stats = {'site': 'google.com', 'traffic': 10000, "type":
"organic"}

>>> print(site_stats.get('site'))

google.com
```

س 74: كيف تجتاز كائن القاموس في بايثون؟

يمكننا استخدام الحلقة "for" و "in" لاجتياز كائن القاموس.

```
>>> site_stats = {'site': 'google.com', 'traffic': 10000, "type":
"organic"}

>>> for k, v in site_stats.items():

    print("The key is: %s" % k)

    print("The value is: %s" % v)

    print("++++++++++++++++++++++++")
```

الإخراج :

```
The key is: type

The value is: organic

++++++++++++++++++++++

The key is: site

The value is: tecbeamers.com

++++++++++++++++++++++

The key is: traffic

The value is: 10000

++++++++++++++++++++++
```

س 75: كيف تضيف عناصر إلى قاموس في بايثون؟

يمكننا إضافة عناصر عن طريق تعديل القاموس بمفتاح جديد ثم تعيين القيمة إليه.

```
>>> # Setup a blank dictionary

>>> site_stats = {}

>>> site_stats['site'] = 'google.com'

>>> site_stats['traffic'] = 10000000000

>>> site_stats['type'] = 'Referral'

>>> print(site_stats)

{'type': 'Referral', 'site': 'google.com', 'traffic': 10000000000}
```

يمكننا حتى الانضمام إلى قواميس للحصول على قاموس أكبر بمساعدة طريقة ()update

```
site_stats = {}

>>> site_stats['site'] = 'google.co.in'

>>> print(site_stats)

{'site': 'google.co.in'}

>>> site_stats_new = {'traffic': 1000000, "type": "social
media"}

>>> site_stats.update(site_stats_new)

>>> print(site_stats)

{'type': 'social media', 'site': 'google.co.in', 'traffic':
1000000}
```

س 76: كيف تحذف عناصر القاموس في بايثون؟

يمكننا حذف مفتاح في القاموس باستخدام طريقة ()del

```
>>> site_stats = {'site': ' google.com', 'traffic': 10000, "type":
"organic"}

>>> del site_stats["type"]

>>> print(site_stats)

{'site': 'google.com', 'traffic': 1000000}
```

طريقة أخرى ، يمكننا استخدامها هي وظيفة ()pop يقبل المفتاح كمعلمة. أيضًا ، معلمة ثانية ، يمكننا تمرير قيمة افتراضية إذا لم يكن المفتاح موجودًا.

```
>>> site_stats = {'site': 'google.com', 'traffic': 10000, "type":
"organic"}

>>> print(site_stats.pop("type", None))

organic

>>> print(site_stats)

{'site': 'google.com', 'traffic': 10000}
```

س 77: كيف تتحقق من وجود مفتاح في القاموس؟

يمكننا استخدام بايثون عامل التشغيل"in" لاختبار وجود مفتاح داخل كائن dict.

```
>>> site_stats = {'site': 'google.com', 'traffic': 10000,
"type": "organic"}

>>> 'site' in site_stats

True

>>> 'traffic' in site_stats

True

>>> "type" in site_stats

True
```

في وقت سابق ، قدمت بايثون أيضًا طريقة has_key() التي تم إهمالها.

س 78: ما هو بناء الجملة لفهم القائمة في بايثون؟

التوقيع على فهم القائمة كما يلي:

```
[ expression(var) for var in iterable ]
```

على سبيل المثال ، سيعيد الرمز أدناه جميع الأرقام من 10 إلى 20 ويخزنها في قائمة.

```
>>> alist = [var for var in range(10, 20)]
>>> print(alist)
```

س 79: ما هو بناء الجملة لفهم القاموس في بايثون؟

يحتوي القاموس على نفس البنية كما كان لفهم القائمة ولكن الفرق هو أنه يستخدم أقواس متعرجة:

```
{ aKey, itsValue for aKey in iterable }
```

على سبيل المثال ، سيعيد الرمز أدناه جميع الأرقام من 10 إلى 20 كمفاتيح وسيخزن المربعات المقابلة لتلك الأرقام كقيم.

```
>>> adict = {var:var**2 for var in range(10, 20)}
>>> print(adict)
```

س 80: ما هي صيغة تعبير المولد في بايثون؟

يتطابق بناء جملة تعبير المولد مع فهم القائمة ، ولكن الفرق هو أنه يستخدم قوسين:

```
( expression(var) for var in iterable )
```

على سبيل المثال ، سيقوم الكود التالي بإنشاء كائن منشئ يولد القيم من 10 إلى 20 عند استخدامه.

```
list((var for var in range(10, 20)))

<generator object <genexpr> at 0x02FA8840>

list((var for var in range(10, 20)))

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

الآن ، انظر المزيد من أسئلة مقابلة بايثون للممارسة.

س 81: كيف تكتب تعبير شرطي في بايثون؟

يمكننا استخدام العبارة الفردية التالية كتعبير شرطي. العبارة الافتراضية إذا كان الشرط آخر بيان آخر

```
>>> no_of_days = 366

>>> is_leap_year = "Yes" if no_of_days == 366 else "No"

>>> print(is_leap_year)

Yes
```

س 82: ماذا تعرف عن تعداد بايثون؟

أثناء استخدام التكرارات ، قد يكون لدينا أحيانًا حالة استخدام لتخزين عدد التكرارات. تحصل بايثون على هذه المهمة بسهولة تامة من خلال إعطاء طريقة مضمنة تُعرف بـ ()enumerate.

تقوم الدالة ()enumerate بإرفاق متغير عداد إلى عنصر قابل للتكرار وإعادته ككائن "enumerated ".

يمكننا استخدام هذا الكائن مباشرة في حلقات "for" أو تحويله إلى قائمة الصفوف من خلال استدعاء طريقة ()list. لديه التوقيع التالي:

```
enumerate(iterable, to_begin=0)
```

```
Arguments:

iterable: array type object which enables iteration

to_begin: the base index for the counter is to get started, its
default value is 0
```

```
# Example - enumerate function

alist = ["apple","mango", "orange"]

astr = "banana"


# Let's set the enumerate objects

list_obj = enumerate(alist)

str_obj = enumerate(astr)


print("list_obj type:", type(list_obj))

print("str_obj type:", type(str_obj))


print(list(enumerate(alist)) )

# Move the starting index to two from zero

print(list(enumerate(astr, 2)))
```

الإخراج :

```
list_obj type: <class 'enumerate'>

str_obj type: <class 'enumerate'>

[(0, 'apple'), (1, 'mango'), (2, 'orange')]

[(2, 'b'), (3, 'a'), (4, 'n'), (5, 'a'), (6, 'n'), (7, 'a')]
```

س 83: ما هو استخدام وظيفة globals() في بايثون ؟

تعرض الدالة globals() في بايثون جدول الرموز العمومية الحالي ككائن قاموس.

تحتفظ بايثون بجدول رموز للاحتفاظ بكل المعلومات الضرورية حول البرنامج. تتضمن هذه المعلومات أسماء المتغيرات والأساليب والفئات التي يستخدمها البرنامج.

تبقى جميع المعلومات الواردة في هذا الجدول في النطاق العالمي للبرنامج ويسمح لنا بايثون باستردادها باستخدام طريقة globals().

```
Signature: globals()


Arguments: None
```

```
# Example: globals() function
x = 9
def fn():
    y = 3
    z = y + x
    # Calling the globals() method
    z = globals()['x'] = z
    return z
```

الإخراج :

```
12
```

س 84: لماذا تستخدم طريقة ()zip في بايثون؟

تتيح لنا طريقة zip تعيين الفهرس المقابل لعدة حاويات حتى نتمكن من استخدامها باستخدام وحدة واحدة.

```
Signature:
 zip(*iterators)
Arguments:
 Python iterables or collections (e.g., list, string, etc.)
Returns:
 A single iterator object with combined mapped values
```

```
# Example: zip() function


emp = [ "tom", "john", "jerry", "jake" ]

age = [ 32, 28, 33, 44 ]

dept = [ 'HR', 'Accounts', 'R&D', 'IT' ]


# call zip() to map values

out = zip(emp, age, dept)


# convert all values for printing them as set

out = set(out)


# Displaying the final values

print ("The output of zip() is : ",end="")

print (out)
```

الإخراج:

```
The output of zip() is : {('jerry', 33, 'R&D'), ('jake', 44, 'IT'),
('john', 28, 'Accounts'), ('tom', 32, 'HR')}
```

س 85: ما هي متغيرات الفئة أو الثابتة في برمجة بايثون ؟

في بايثون ، تشترك جميع الكائنات في فئة مشتركة أو متغيرات ثابتة.

لكن المتغيرات المثيلة أو غير الثابتة مختلفة تمامًا لأشياء مختلفة.

تحتاج لغات البرمجة مثل C++ و Java إلى استخدام الكلمة الأساسية الثابتة لإنشاء متغير كمتغيرclss. ومع ذلك ، فإن بايثون لديها طريقة فريدة لتعريف متغير ثابت.

تصبح جميع الأسماء التي تمت تهيئتها بقيمة في إعلان class متغيرات class. وتلك التي تحصل على قيم معينة في أساليب الصنف تصبح متغيرات الحالة.

```
# Example
class Test:
    aclass = 'programming' # A class variable
    def __init__(self, ainst):
        self.ainst = ainst # An instance variable

# Objects of CSStudent class
test1 = Test(1)
test2 = Test(2)

print(test1.aclass)
print(test2.aclass)
print(test1.ainst)
print(test2.ainst)
# A class variable is also accessible using the class name
print(Test.aclass)
```

الإخراج:

```
programming

programming

1

2

programming
```

دعنا الآن نجيب على بعض أسئلة مقابلة بايثون المتقدمة المستوى.

س 86: كيف يعمل العامل الثلاثي في بايثون؟

عامل الثلاثية هو بديل للعبارات الشرطية. فهو يجمع بين القيم الحقيقية أو الكاذبة ببيان تحتاج إلى اختباره.

تبدو الصيغة مثل تلك الواردة أدناه.

[onTrue] if [Condition] else [onFalse]

```
x, y = 35, 75

smaller = x if x < y else y

print(smaller)
```

س -87: ماذا تفعل الكلمة "self"؟

self هي كلمة أساسية في بايثون تمثل متغيرًا يحتفظ بمثيل كائن.

في جميع اللغات الموجهة للكائنات تقريبًا ، يتم تمريرها إلى الأساليب كمعلمة مخفية.

س 88: ما هي الطرق المختلفة لنسخ كائن في بايثون؟

هناك طريقتان لنسخ الكائنات في بايثون .

- وظيفة copy.copy()
  - يجعل نسخة من الملف من المصدر إلى الوجهة.
  - ستعرض نسخة سطحية من المعلمة.
- وظيفة copy.deepcopy()
  - كما ينتج نسخة من كائن من المصدر إلى الوجهة.
  - ستعرض نسخة عميقة من المعلمة التي يمكنك تمريرها إلى الوظيفة.

س 89: ما غرض التوثيق في بايثون؟

في بايثون docstring، هو ما نسميه الوثائق. يقوم بتعيين عملية تسجيل وظائف بايثون modules, and classes.

س 90: ما هي وظيفة بايثون التي ستستخدمها لتحويل رقم إلى سلسلة؟

لتحويل رقم إلى سلسلة ، يمكنك استخدام str() الدالة المضمنة. إذا كنت تريد تمثيلًا ثمانيًا أو سداسيًا عشريًا ، فاستخدم الدالة المضمنة oct() أو hex().

س 91: كيف تقوم بتصحيح برنامج في بايثون؟ هل من الممكن أن تتصفح كود بايثون؟

نعم ، يمكننا استخدام مصحح أخطاء بايثون لتصحيح أي برنامج بايثون . وإذا بدأنا برنامجًا باستخدام pdb ، فسيسمح لنا حتى بالتدقيق في التعليمات البرمجية.

س 92: قم بإدراج بعض أوامر PDB لتصحيح أخطاء برامج بايثون ؟

فيما يلي بعض أوامر PDB لبدء تصحيح كود بايثون .

- إضافة نقطة توقف (B)
- استئناف التنفيذ (C)
- التصحيح خطوة بخطوة (S)
- الانتقال إلى السطر التالي (n)
- قائمة كود المصدر (l)
- طباعة تعبير (P)

س 93: ما هو الأمر لتصحيح برنامج بايثون؟

يساعد الأمر التالي في تشغيل برنامج بايثون في وضع التصحيح

```
$ python -m pdb python-script.py
```

س 94: كيف تراقب تدفق الكود لبرنامج في بايثون؟

في بايثون يمكننا استخدام طريقة ضبط (sys) للوحدة النمطية sys لإعداد خطاطيف التتبع ومراقبة الوظائف داخل البرنامج. تحتاج إلى تحديد طريقة رد اتصال تتبع وتمريرها إلى الدالة ()settrace.يجب أن يحدد رد الاتصال ثلاث حجج كما هو موضح أدناه.

```
import sys

def trace_calls(frame, event, arg):

    # The 'call' event occurs before a function gets executed.

    if event != 'call':

        return

    # Next, inspect the frame data and print information.

    print 'Function name=%s, line num=%s' %
(frame.f_code.co_name, frame.f_lineno)

    return

def demo2():

    print 'in demo2()'

def demo1():

    print 'in demo1()'

    demo2()

sys.settrace(trace_calls)

demo1()
```

س 95: لماذا ومتى تستخدم المولدات في بايثون؟

مولد في بايثون هو دالة تقوم بإرجاع كائن قابل للتكرار. يمكننا التكرار على كائن المولد باستخدام yield keyword. لكن يمكننا القيام بذلك مرة واحدة فقط لأن قيمهم لا تستمر في الذاكرة ، بل تحصل على القيم بسرعة.

تمنحنا المولدات القدرة على الاستمرار في تنفيذ دالة أو خطوة طالما أردنا الاحتفاظ بها. ومع ذلك ، إليك بعض الأمثلة حيث من المفيد استخدام المولدات.

- يمكننا استبدال الحلقات بمولدات لحساب النتائج التي تتضمن مجموعات بيانات كبيرة بكفاءة.
- تكون المولدات مفيدة عندما لا نريد جميع النتائج ونرغب في التراجع لبعض الوقت.
- بدلاً من استخدام وظيفة رد الاتصال ، يمكننا استبدالها بمولد. يمكننا كتابة حلقة داخل الوظيفة تفعل نفس الشيء مثل رد الاتصال وتحويله إلى مولد.

س 96: ماذا يفعل yield keyword تفعل في بايثون ؟

يمكن للكلمة الرئيسية yield المحصلة تحويل أي وظيفة إلى مولد. يعمل مثل كلمة رئيسية قياسية للعودة. لكنها ستعيد دائمًا كائن المولد. أيضًا ، يمكن أن تحتوي الطريقة على مكالمات متعددة للكلمة yield الرئيسية.

انظر المثال أدناه.

```
def testgen(index):
  weekdays = ['sun','mon','tue','wed','thu','fri','sat']
  yield weekdays[index]
  yield weekdays[index+1]

day = testgen(0)
print (next(day), next(day))

#output: sun mon
```

س 97: كيفية تحويل قائمة إلى أنواع بيانات أخرى؟

في بعض الأحيان ، لا نستخدم القوائم كما هي. بدلاً من ذلك ، علينا تحويلها إلى أنواع أخرى.

تحويل قائمة إلى سلسلة.

يمكننا استخدام طريقة. ()join التي تجمع كل العناصر في عنصر واحد وترجع كسلسلة.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat']

listAsString = ' '.join(weekdays)

print(listAsString)


#output: sun mon tue wed thu fri sat
```

تحويل قائمة إلى مجموعة.

استدعاء دالة ()tuple في بايثون لتحويل قائمة إلى tuple.

هذه الوظيفة تأخذ القائمة كوسيطة لها.

لكن تذكر ، لا يمكننا تغيير القائمة بعد تحويلها إلى مجموعة لأنها تصبح ثابتة.

```
weekdays = ['sun','mon','tue','wed','thu','fri','sat']

listAsTuple = tuple(weekdays)

print(listAsTuple)


#output: ('sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat')
```

تحويل قائمة إلى مجموعة.

تحويل قائمة إلى مجموعة له آثار جانبية.

- لا يسمح التعيين بإدخالات مكررة بحيث يزيل التحويل أي عنصر من هذا القبيل.
- المجموعة عبارة عن مجموعة مرتبة ، لذلك سيتغير ترتيب عناصر القائمة أيضًا.

ومع ذلك ، يمكننا استخدام الدالة ()set لتحويل قائمة إلى مجموعة.

```
weekdays =
['sun','mon','tue','wed','thu','fri','sat','sun','tue']

listAsSet = set(weekdays)

print(listAsSet)



#output: set(['wed', 'sun', 'thu', 'tue', 'mon', 'fri',
'sat'])
```

تحويل قائمة إلى قاموس.

في القاموس ، يمثل كل عنصر زوجًا بقيمة مفتاح. لذا فإن تحويل القائمة ليس بهذه البساطة كما كان بالنسبة لأنواع البيانات الأخرى.

ومع ذلك ، يمكننا تحقيق التحويل عن طريق تقسيم القائمة إلى مجموعة من الأزواج ثم استدعاء الدالة ()zip لإعادتها كمجموعات.

سيؤدي تمرير الصفوف إلى دالة ()dict إلى تحويلها أخيرًا إلى قاموس.

```
weekdays = ['sun','mon','tue','wed','thu','fri']

listAsDict = dict(zip(weekdays[0::2], weekdays[1::2]))

print(listAsDict)

#output: {'sun': 'mon', 'thu': 'fri', 'tue': 'wed'}
```

س 98: كيف تحسب تكرارات كل عنصر موجود في القائمة دون ذكرها صراحة؟

على عكس المجموعات ، يمكن أن تحتوي القوائم على عناصر لها نفس القيم.

في بايثون، تحتوي القائمة على دالة ()count تقوم بإرجاع تكرارات عنصر معين.

عد تكرارات عنصر فردي.

```
weekdays =
['sun','mon','tue','wed','thu','fri','sun','mon','mon']

print(weekdays.count('mon'))



#output: 3
```

عد تكرارات كل عنصر في القائمة.

سنستخدم قائمة الفهم مع طريقة ()count. ستطبع تكرار كل عنصر.

```
weekdays =
['sun','mon','tue','wed','thu','fri','sun','mon','mon']

print([[x,weekdays.count(x)] for x in set(weekdays)])



#output: [['wed', 1], ['sun', 2], ['thu', 1], ['tue', 1],
['mon', 3], ['fri', 1]]
```

س 99: ما هو NumPy وكيف هو أفضل من قائمة في بايثون؟

NumPy هي حزمة بايثون للحوسبة العلمية يمكنها التعامل مع أحجام البيانات الكبيرة. يتضمن كائن صفيف N-dimensional قوي ومجموعة من الوظائف المتقدمة.

كما أن صفائف NumPy تتفوق على القوائم المضمنة. لا يوجد. من أسباب ذلك.

- صفيفات NumPy مضغوطة أكثر من القوائم.
- قراءة وكتابة العناصر أسرع مع NumPy.
- يعد استخدام NumPy أكثر ملاءمة من القائمة القياسية.
- تعد صفيفات NumPy أكثر كفاءة لأنها تزيد من وظيفة القوائم في بايثون .

س 100: ما هي الطرق المختلفة لإنشاء مصفوفة NumPy فارغة في بايثون ؟

هناك طريقتان يمكننا تطبيقهما لإنشاء صفائف NumPy فارغة.

الطريقة الأولى لإنشاء صفيف فارغ.

```
import numpy

numpy.array([])
```

الطريقة الثانية لإنشاء صفيف فارغ.

```
 # Make an empty NumPy array

 numpy.empty(shape=(0,0))
```

موجز - أسئلة في بايثون الأساسية

نأمل أن يعجب جميعكم بأحدث مجموعة من أسئلة بايثون .

أتمنى يعجبكم الكتاب وأتمنى ان تشاركونا اقتراحاتكم على حسابي الخاص

black-shadow0020@hotmail.com

وتابعوني على قنواتي في تلجرام
https://t.me/shadow YE

https://t.me/SHADOW YEMEN

https://t.me/Shadow books

كان معكم اخوكم رشيد القاضي.