Python Language

كورسات صوت الشسيح
القس يا كوبوس بدري

Python Programming Language

HUAWEI

Python Language

كورسات صوت الشسيح
القس يا كوبوس بدري

Contents

HUAWEI

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Introduction to Python Variables

What is a Variable in Python?

المتغير في بايثون هو اسم رمزي لقيمة.
يتيح لك تخزين البيانات وتحديثها واستخدامها.
المتغيرات في بايثون ذات نوع ديناميكي ولا تتطلب إعلانًا مسبقًا.

- Example: x = 10
- name = 'Alice'
- is_valid = True

Rules for Naming Variables

يجب أن يبدأ بحرف أو شرطة سفلية (_).
يمكن أن يحتوي على أحرف وأرقام وشرطات سفلية.
بايثون حساس لحالة الأحرف: المتغير myVar مختلف عن myvar

- Cannot use reserved keywords (e.g., 'for', 'if').

- Examples: valid_variable = 10
- _underscore_var = 'Python‘
- Invalid: 123number = 789, for = 'loop'

HUAWEI

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

Variable Assignment and Initialization

- Variables are assigned using the '=' operator.
- Data types are determined by the value assigned.
- Python is dynamically typed.

- Example: age = 25
- name = 'John'
- pi = 3.14159
- is_active = True

Multiple Assignment in Python

- Assign values to multiple variables in a single line:
a, b, c = 5, 'Hello', True
- Assign the same value to multiple variables:
x = y = z = 100

Types of Data in Python Variables

- Python supports various data types:
  - Integers (int)
  - Floats (float)
  - Strings (str)
  - Booleans (bool)
  - None (NoneType)

- Examples:

```
x = 10  # int
pi = 3.14  # float
name = 'John'  # str
is_valid = False  # bool
empty = None  # NoneType
```

HUAWEI

Updating Variables

- Variables can be reassigned and even change types.
- Example:

```
count = 5
count = count + 2  # Reassign
count = 'seven'  # Now a string
```

Checking Variable Type with type()

- Use the 'type()' function to check the variable type.
- Example:

```
x = 100
print(type(x))  # Output: <class 'int'>
name = 'Alice'
print(type(name))  # Output: <class 'str'>
```

Dynamic Typing in Python

- Python is dynamically typed; no need to declare types.
- The type is determined at runtime.
- Example:

```
x = 42  # int
x = 'forty-two'  # Now x is a string
```

HUAWEI

Python Language

كورسات صوت الشسيح
القس يا كوبوس بدري

Arithmetic Operations in Python

- Python supports basic arithmetic operations:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)

- Example: a = 10
- b = 3
- result_add = a + b # 13
- result_sub = a - b # 7
- result_mul = a * b # 30
- result_div = a / b # 3.3333

More Arithmetic Operations

- Additional operators:
- Modulus (%)
- Example:

```
a = 10
b = 3
result_mod = a % b  # 1
```

HUAWEI

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Lists

القائمة في بايثون هي مجموعة مرتبة من العناصر يمكنك إضافة عناصر إليها أو حذفها في أي وقت.
يمكن أن تكون أنواع العناصر داخل القائمة مختلفة.
يمكنك الوصول إلى كل عنصر في القائمة باستخدام الفهرس. يبدأ أول فهرس من الرقم 0، وآخر عنصر يمكن الوصول إليه باستخدام الفهرس العكسي -1

Common Operations on Lists

- Access
- Update (append and insert)
- Delete elements (del)
- Operator on list script (+/*)
- List interception

Functions of Python Lists

- cmp(list1, list2),
- len(list)
- max(list)
- min(list)
- list(seq)

HUAWEI


Methods of Python Listing

- list.append(obj)
- list.count(obj)
- list.extend(seq)
- list.index(obj)
- list.insert(index, obj)
- list.pop(obj=list[-1])
- list.remove(obj)
- list.sort([func])
- list.reverse()

HUAWEI

Python Language

كورسات صوت النسيج
القس يا كوبوس بدري

- **Understand the difference between "append" and "extend".**

```
>>>x = [1, 2, 3]
>>>y = [4, 5]
>>>x.append(y)
>>>print(x) [1,2,3,[4,5]]
```

```
>>>x = [1, 2, 3]
>>>y = [4, 5]
>>>x.extend(y) # equal to
>>>for i in y:
>>>x.append(i)
>>>print(x) [1,2,3,4,5]
```

- Copy a list.

```
#Method one:
new_list = old_list[:]
#Method two:
new_list = list(old_list)
```

- Check whether the list is empty.

```
if len(items) == 0:
    print("empty list") #or
if items == []:
   print("empty list")
```

Get last element of a list

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> a[len(a)-1] 10
>>> a[-1]
10
```

HUAWEI

- **Remove elements from a list.**

```
>>> a = [0, 2, 2, 3]
>>> a.remove(2)
# ValueError will be returned if the removed element is not within
the list.
>>> a.remove(7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list·
```

"del" removes a certain element in a specified position.

"pop" is similar to "del", but "pop" can return the removed element.

- Connect two lists.

```
>>>listone = [1, 2, 3]
>>>listtwo = [4, 5, 6]
>>>mergedlist = listone + listtwo
>>>print(mergelist)
[1, 2, 3, 4, 5, 6]
```

HUAWEI

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Tuples

هي نوع من أنواع البيانات في لغة **بايثون** تُستخدم لتخزين مجموعة من العناصر **Tuples** إلى حد كبير، ولكنها تختلف عنها في **(Lists**(قيم متعددة) في كائن واحد. تشبه القوائم (، أي لا يمكن تعديل محتوياتها بعد إنشائها.**(Immutable**كونها **غير قابلة للتغيير (**

خصائص الـ
Tuples
غير قابلة للتغيير
.Tupleلا يمكنك إضافة أو حذف عناصر بعد إنشاء الـ
تسمح بتكرار العناصر:
يمكن أن تحتوي على عناصر مكررة.
تدعم الأنواع المختلفة:
يمكن أن تحتوي على بيانات من أنواع مختلفة (مثل أرقام، نصوص، قوائم، وغيرها).
ترتيب العناصر داخله يظل كما هو.

كيفية إنشاء Tuple

```
# إنشاءTuple
my_tuple = (1, 2, 3, "Hello", True)
print(my_tuple)
```

الوصول إلى العناصر
يمكنك الوصول إلى العناصر باستخدام **الفهارس (Indexes)**.

```
my_tuple = (10, 20, 30, 40)

# الوصول إلى العنصر الأول
print(my_tuple[0])  # الناتج: 10

# الوصول إلى العنصر الأخير
print(my_tuple[-1])  # الناتج: 40
```

لماذا نستخدمه؟
الأداء(Performance)
.أسرع من القوائم لأنها غير قابلة للتغيير
:الحماية
Tuplesإذا كنت لا تريد تغيير البيانات، يمكنك استخدام
لضمان الثبات

Tuplesمتى تستخدم؟
عند التعامل مع بيانات ثابتة لا تحتاج إلى تعديل.
عند الحاجة إلى الأداء السريع مقارنة بالقوائم.

Python Language

كورسات صوت النسيج
القس باكوبوس بدري

if Statements (شروط)

- Python يدعم ثلاثة هياكل التحكم: if, for, and while.
- In Python, if statements تستخدم للتحكم في تنفيذ برامج التحكم،

```
if الشرط الاول:
        الحاله الاولى... Judging
elif condition 2:  Statement 2…
        Judging condition 3:
elif  Statement 3…

else: Statement 4…
```

والشكل الأساسي هو:

مثال : لعبة التنين

```
print("أحدهما يحتوي على تنين ودود والآخر به تنين جائع .أمامك كهفان!")
choice = input("اختر الكهف (1 أو 2): ")

# التحقق من الكهف الذي اختاره اللاعب
if choice == "1":
    print("تلتقي بتنين ودود يشاركك كنزه!")
else:
    print("اللعبة انتهت .التنين الجائع يلتهمك!")
```

Python Language

كورسات صوت الشيح
القس ياكوبوس بدري

while Statements

- The while statement in the Python language

. يُستخدم لتنفيذ برنامج تكراري، والذي،في ظل ظروف معينة، يتكرر عبر برنامج للتعامل مع نفس المهام التي تحتاج إلى التكرار.

- (Break)عندما يكون الشرط حقيقي، لن تنتهي الحلقة أبدًا، وتشكل حلقة لا نهائية، تُعرف أيضًا باسم الحلقة الميتة. يمكنك استخدام عبارة في حلقة لإجبار حلقة ميتة على الانتهاء.

- How to use a while statement:

```
count = 0

while (count < 9):
    print("The count is:", count) count =
    count + 1

print("Good bye!")
```

for Statements

- In the Python language, the for loop can traverse any items of a sequence , such as a list, a dictionary, or a string.
- The for statement is different from a traditional for statement. The former accepts an iterative object (such as a sequence or iterator) as its argument, and one element is iterated each time.

```
for num in nums: if num == 1:
        print(num+"---")
    elif num == 2:
        print(num+"///")
    else:
        print('break not triggered')
```

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

for Statements Con…

- Example:

```
>>> for i in range(0,10):
>>>print(i)
```

- Output:

```
0
1
2
3
4
5
6
7
8
9
```

- Example:

```
>>> a=[1,3,5,7,9]
>>> for i in a:
>>>print(i)
```

- Output:

```
1
3
5
7
9
```

Python Language

كورسات صوت الشيح
القس ياكوبوس بدري

for Statements Con…

لعبة تخمين الرقم

```
import random

# إنشاء رقم عشوائي بين 1 و100
secret_number = random.randint(1, 100)

print("مرحبًا بك في لعبة تخمين الرقم!")
print("حاول تخمين رقم بين 1 و100.")

# حلقة تستمر حتى يخمن اللاعب الرقم الصحيح
while True:
    guess = int(input("أدخل تخمينك: "))
    if guess < secret_number:
        print("الرقم أقل! حاول مرة أخرى.")
    elif guess > secret_number:
        print("الرقم أكبر! حاول مرة أخرى.")
    else:
        print(f"مبروك! لقد خمنت الرقم الصحيح: {secret_number}.")
        break
```

Loop Nesting

- Python allows one loop to be nested in another loop.
- Syntax of for loop nesting:

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statement(s)
    statement(s)
```

- Syntax of while loop nesting:

```
while expression:
    while expression:
        statement(s)
    statement(s)
```

break and continue

- **A break statement ends the entire loop, and if a break statement is triggered, the loop else is not executed**
  (ينهي الحلقة بأكملها)
- **A continue statement ends the ongoing iteration of the loop, and begins the next iteration.**

  (ينهي التكرار المستمر للحلقة، ويبدأ التكرار التالي)
- **If you use a nested loop, the break statement stops executing the deepest loop and starts executing the next line of code.**
- **The continue statement tells Python to skip the remaining statements of the current loop to proceed to the next round of loops.**
- **Both the break and continue statements are available in the while and for loops.**

Python Language

كورسات صوت الشسيح
القس ياكوبوس بدري

Python Functions(الدوال)

الدالة هي جزء من الكود يتم تنظيمه وإعادة استخدامه، ويستخدم لتنفيذ وظيفة واحدة أو وظائف مرتبطة.

يمكن أن تحسن الدوال من هيكلية التطبيقات وقابلية إعادة استخدام الكود.

يمكنك أيضًا إنشاء دوال خاصة بك، والتي تسمى الدوال التي يُعرفها المستخدم.

Common built-in functions(مدعمه من اللغه نفسها)

```
>>> int('123')
123
>>> int(12.34)
12
>>> float('12.34')
12.34
>>> str(1.23)
'1.23'
>>> str(100)
'100'
>>> bool(1)
True
>>> bool('')
False
```

Defining a Function Con…

- Example:

```
def my_abs(x):
    if not isinstance(x, (int, float)):
        raise TypeError('bad operand type')
    if x >= 0:
        return x
    else:
        return -x
```

Arqumen
ts

Calling a Function(الاستدعاء)

تعريف الدالة يعطي الدالة اسمًا فقط، ويحدد المعاملات (Arquments)الموجودة في الدالة، وبنية كتلة الكود.

بعد إتمام الهيكل الأساسي لهذه الدالة، يمكنك تنفيذها من خلال استدعاء دالة أخرى، أو يمكنك تنفيذها مباشرة من موجه بايثون.

```
# Define a function def test(str):
    print(str)
    return str
# Call a function
test("I want to call a user-defined function!")
```

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Transferring Arguments

```
def greet(name, age):
    print(f"Hello, my name is {name} and I am {age} years old.")
# استدعاء الدالة مع المعاملات الأساسية
greet("Ali", 25)

def greet(name, age):
    print(f"Hello, my name is {name} and I am {age} years old."
# استدعاء الدالة باستخدام المعاملات بالاسم
greet(age=30, name="Ahmed")

def greet(name, age=18):
    print(f"Hello, my name is {name} and I am {age} years old.")
# استدعاء الدالة مع المعامل الافتراضي
greet("Fatima")  # سيتم استخدام العمر الافتراضي

def print_names(*names):
    for name in names:
        print(name)
# استدعاء الدالة مع عدد غير محدد من المعاملات
print_names("Ali", "Sara", "Khalid", "Layla")
```

Argument Types

فيما يلي أنواع المعاملات الرسمية التي يمكنك استخدامها عند استدعاء دالة بايثون:

المعاملات الأساسية: يجب تمرير المعاملات الأساسية إلى الدالة بالترتيب الصحيح،ويجب أن يكون عدد المعاملات عند الاستدعاء هو نفسه كما تم تعريف

المعاملات بالاسم: المعاملات بالاسم والدوال يتم استدعاؤها معًا عن كثب، ويستخدم استدعاء الدالة المعاملات بالاسم لتحديد قيم المعاملات الوار

المعاملات الافتراضية: عند استدعاء دالة، إذا لم يتم تمرير قيمة للمعامل الافتراضي، يعتبر أنه تم استخدام القيمة الافتراضية.

المعاملات ذات الطول غير المحدود: قد تحتاج الدالة إلى التعامل مع عدد أكبر من المعاملات مقارنة بما تم تحديده أصلاً. تسمى هذه المعاملات بالمعاملات غير المحدودة ولا يتم تسميتها عند تحديدها.

Python Language

كورسات صوت الشسيح
القس يا كوبوس بدري

Anonymous Functions

- Syntax of lambda function:

```
lambda arguments: expression
```

- Example:

```
# Program to show the use of lambda functions
double = lambda x: x * 2

print(double(5))
```

- Output:

```
10
```

Global Variables and Local Variables

المتغير الذي يتم تعريفه داخل وظيفة (function) يكون له نطاق محلي ويُسمى **متغير محلي** (local variable)، بينما المتغير الذي يتم تعريفه خارج الوظيفة يكون له نطاق عام ويُسمى **متغير عام** (global variable).

يمكن الوصول إلى المتغيرات المحلية فقط داخل الوظيفة التي تم تعريفها فيها، بينما تكون المتغيرات العامة متاحة في جميع أجزاء البرنامج.

عندما يتم استدعاء وظيفة، يتم إضافة جميع أسماء المتغيرات المُعرّفة داخل الوظيفة إلى النطاق.

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Object-Oriented Programming

البرمجة كائنية التوجه (Object-Oriented Programming - OOP) هي فلسفة تصميم البرامج. تعتبر OOP الكائنات الوحدات الأساسية للبرنامج، حيث يحتوي الكائن على بيانات ووظائف لمعالجة هذه البيانات.
في المقابل، البرمجة المعتمدة على العمليات (Process-Oriented Programming) تعتبر البرنامج سلسلة من مجموعات الأوامر، أي التنفيذ المتسلسل لمجموعة من الوظائف. ولتبسيط تصميم البرنامج، تقوم البرمجة المعتمدة على العمليات بتقسيم الوظائف إلى وظائف فرعية، مما يقلل من تعقيد النظام من خلال تقسيم الوظائف الكبيرة إلى وظائف أصغر.

تعامل OOP البرامج الحاسوبية كمجموعة من الكائنات، حيث يمكن لكل كائن استقبال رسائل من كائنات أخرى ومعالجتها. ويتم تنفيذ البرنامج كعملية تمرير رسائل بين الكائنات.

في لغة Python، يمكن اعتبار جميع أنواع البيانات ككائنات، ويمكن تخصيص الكائنات.
نوع بيانات الكائن المخصص هو المفهوم المعروف بـ **الفئة (Class)** في البرمجة كائنية التوجه.

Common Python OOP Terms

- التجريد / التنفيذ (Abstract/Implementation)
- التغليف / الواجهة (Encapsulation/Interface)
- التكوين (Composition)
- الاشتقاق / الوراثة / هيكل الوراثة (Derivation/Inheritance/Inheritance Structure)
- التعميم / التخصيص (Generalization/Specialization)
- التعددية الشكلية (Polymorphism)
- التأمل / الانعكاس (Introspection/Reflection)

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Classes(Objects or كائنات)

الفئة (Class) هي هيكل بيانات يُستخدم لتعريف الكائنات التي تجمع بين قيم البيانات والخصائص السلوكية. الفئة هي كيان مجرد من العالم الحقيقي يظهر بطريقة برمجية، ويتم تجسيد هذه الكائنات من خلال إنشاء نسخ (Instances). كتشبيه، الفئة هي بمثابة مخطط أو نموذج يُستخدم لإنتاج كائنات حقيقية (النسخ).

في لغة Python، يشبه تعريف الفئة تعريف الدالة، حيث تبدأ الكلمة المفتاحية في السطر الأول، متبوعة بجسم الكود الذي يمثل تعريفها، كما يلي:

```
def  functionName(args):  ‘
function documentation string'
   function_suite

class ClassName(object):
   ‘Click class documentation string’
   class_suite
```

```
class Dog:
    """a simple try of simulating a dog"""
    def __init__(self,name,age):
        """Initializeattribute: name and age"""
        self.name = name
        self.age = age
    def sit(self):
        """Simulate sitting when a dog is ordered to do so"""
        print(self.name.title()+" is now sitting")
    def roll_over(self):
        """Simulate rolling over when a dog is ordered to do so"""
        print(self.name.title()+" rolled over!")
```

```
dog = Dog('Max',3)
dog.sit()
```

```
Max is now sitting
```

Python Language

كورسات صوت الشسيح
القس يا كوبوس بدري

Classes con...

دالة `__init__()` تُستدعى تلقائيًا في كل مرة يتم فيها استخدام الفئة لإنشاء كائن جديد.

يُستخدم `__init__()` لتعيين قيم لخصائص الكائن أو تنفيذ العمليات الضرورية عند إنشاء الكائن.

معامل `self` هو مرجع يشير إلى النسخة الحالية من الفئة، ويُستخدم للوصول إلى المتغيرات التي تنتمي إلى الفئة.

Inheritance

الوراثة هي طريقة لإنشاء فئة. في Python، يمكن للفئة أن ترث من واحدة أو أكثر من الفئات الأصلية. تُسمى الفئة الأصلية بالفئة الأساسية أو الفئة العليا (Base Class أو Superclass).

إذا كان هناك عدة فئات تحتوي على سمات متغيرة وخصائص دوال مشتركة، يمكن استخراج هذه السمات والخصائص كخصائص للفئة الأساسية. يتم تعريف السمات المتغيرة والخصائص المميزة في هذه الفئة بحيث تكون خصائص الفئة الأساسية قابلة للوصول فقط عند وراثة الفئة الأساسية. هذا يزيد من قابلية تطوير الكود.

التجريد هو استخراج الأجزاء المتشابهة.

الفئة الأساسية هي فئة تُجرد الخصائص المشتركة بين عدة فئات.

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

Inheritance con…

- Example:

```
Student name is Ahmed
Age is 10
School name is ABC
```

```
151 class person:
152     def __init__(self, **kw):
153         self.name = kw["name"]
154         self.age = kw["age"]
155     def printPerson(self):
156         print('Student name is',self.name)
157         print('Age is',self.age)
158 
159 class student(person):
160     def __init__(self,schoolname,**kw):
161         person.__init__(self,**kw)
162         self.schoolname=schoolname
163     def printStudent(self):
164         self.printPerson()
165         print('School name is',self.schoolname)
166 
167 student1= student("ABC", name= 'Ahmed',age=10)
168 student1.printStudent()
```

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

Sub Classes

أحد الجوانب القوية في البرمجة كائنية التوجه (OOP) هو القدرة على استخدام فئة محددة جيدًا، وتوسيعها أو تعديلها دون التأثير على أجزاء أخرى من الكود التي تستخدم الفئات الموجودة في النظام. يسمح التصميم كائني التوجه (OOD) بوراثة ميزات الفئة بواسطة الفئة التابعة أو الفئة الفرعية. ترث هذه الفئات الفرعية خصائصها الأساسية من الفئة الأساسية (أو الفئة الأسلاف، الفئة العليا). كما يمكن تمديد هذه الاشتقاقات إلى أجيال متعددة.

الفئات المرتبطة في اشتقاق هرمي (أو المتجاورة عموديًا في مخطط شجرة الفئات) تكون في علاقة أبوة وفئة فرعية. أما الفئات التي تشتق من نفس الفئة الأساسية (أو المتجاورة أفقيًا في مخطط شجرة الفئات) تكون في علاقة أشقاء. تُعتبر الفئة الأصلية وجميع الفئات ذات المستوى الأعلى أسلافًا.

Privatization

- **الوصول الافتراضي:**
  تكون الخصائص في Python افتراضيًا "عامة" (Public) ويمكن الوصول إليها من داخل الوحدة التي تحتوي على الفئة وأيضًا من الوحدات الأخرى التي تستورد هذه الوحدة.
- **اللغات الكائنية الأخرى:**
  العديد من لغات البرمجة الكائنية تضيف إمكانية التحكم في مدى رؤية البيانات، بحيث توفر دوال للوصول إلى قيمها فقط.
- **وصول الخصائص والأعضاء:**
  معظم لغات البرمجة الكائنية تقدم رموزًا للتحكم في وصول الأعضاء (مثل الخصائص والدوال).
- **شرطة سفلية مزدوجة (`__`):**
  تقدم Python شكلاً أوليًا للخصوصية لعناصر الفئة (الخصائص والدوال). الخصائص التي تبدأ بشرطة سفلية مزدوجة تعتبر "مربكة" وقت التشغيل، وبالتالي لا يُسمح بالوصول المباشر إليها. في الواقع، يتم إعادة تسمية الخاصية بإضافة شرطة سفلية واسم الفئة في بدايتها.
- **شرطة سفلية واحدة (`_`):**
  للخصوصية البسيطة على مستوى الوحدة، يمكنك استخدام شرطة سفلية واحدة قبل اسم الخاصية. يمنع ذلك تحميل الخصائص الخاصة بالوحدة عند استخدام الصيغة `"* from mymodule import"`. يعتمد هذا الأسلوب على النطاق فقط، وينطبق أيضًا على الدوال.

Python Language

كورسات صوت النسيح
القس يا كوبوس بدري

Privatization

```
line 169, in <module>
    print(student1.__age)
```

ERROR

```
AttributeError: 'student' object has
no attribute '__age'
```

_name is protected attribute
__age is a private attribute

```
151 class person:
152     def __init__(self, **kw):
153         self._name = kw["name"]
154         self.__age = kw["age"]
155     def printPerson(self):
156         print('Student name is',self._name)
157         print('Age is',self.__age)
158
159 class student(person):
160     def __init__(self,schoolname,**kw):
161         person.__init__(self,**kw)
162         self.schoolname=schoolname
163     def printStudent(self):
164         self.printPerson()
165         print('School name is',self.schoolname)
166
167 student1= student("ABC", name= 'Ahmed',age=10)
168 student1.printStudent()
169 print(student1.__age)
```

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

Abstract Class

```
from abc import ABC, abstractmethod

class Animal(ABC):  # صف مجرد
    @abstractmethod
    def make_sound(self):
        pass  # تعريف بدون تنفيذ

class Dog(Animal):  # صف فرعي
    def make_sound(self):
        return "Woof!"

# animal = Animal()  # خطأ: لا يمكن إنشاء كائن من صف مجرد
dog = Dog()        # مسموح لأن Dog توفر تنفيذ الطريقة
print(dog.make_sound())  # "Woof!"
```

الـ `abstract class` هو **صف لا يمكن إنشاء كائنات منه مباشرة**، ويُستخدم كقالب (Template) لفرض قواعد على الصفوف التي ترث منه.

الخصائص:

1. يحتوي على طرق (Methods) **مجردة** بدون تنفيذ.
2. يمكن أن يحتوي على طرق عادية (مع تنفيذ).
3. الصفوف الفرعية (Derived Classes) **ملزمة** بتنفيذ الطرق المجردة.

الاستخدام:

- يُستخدم لتوحيد البنية بين الصفوف الموروثة.
- يُساعد في تصميم الكود بطريقة منظمة وقابلة للتوسيع.

كيفية إنشاء صف مجرد:

1. استيراد مكتبة `abc`.
2. استخدام `ABC` كصف أساس.
3. تعريف الطرق المجردة باستخدام `@abstractmethod`.


Dictionaries

في بايثون هي نوع بيانات يُستخدم لتخزين البيانات في شكل **مفتاح: قيمة (Key: Value)**.
وتسمح (**mutable**)تُعتبر الـ غير مرتبة (في الإصدارات القديمة قبل 3.7) لكنها قابلة للتعديل بأن تكون المفاتيح فريدة.

```
# مثال على إنشاء Dictionary
person = {
    "name": "Ali",  # key:value
    "age": 25,
    "city": "Cairo"
}
print(person)
```

Output

```
{'name': 'Ali', 'age': 25, 'city': 'Cairo'}
```

الوصول إلى القيم
بالاعتماد على المفتاح

```
print(person["name"])  # طباعة قيمة المفتاح 'name'
print(person["age"])   # طباعة قيمة المفتاح 'age'
```

Output

```
Ali
25
```

 # get() تُعيد القيمة إذا كان المفتاح موجودًا، أو القيمة الافتراضية إذا لم يكن موجودًا

```
print(person.get("name"))    # Ali
print(person.get("gender", "Not specified"))  # Not specified
```

Python Language

كورسات صوت الشيح
القس باكوبوس بدري

إضافة وتعديل العناصر

اضافة عنصر

```
person["gender"] = "Male"
print(person)
```

عرض المفاتيح

```
person = {"name": "Ali", "age": 25, "city": "Cairo"}
for key in person:
    print(key)
```

تعديل قيمة موجودة

```
person["age"] = 30
print(person)
```

عرض القيم

```
for value in person.values():
    print(value)
```

حذف باستخدام pop()

```
gender = person.pop("gender")
print(person)  # القاموس بعد الحذف
print(gender)  # العنصر المحذوف
```

عرض القيم والمفاتيح معا

```
for key, value in person.items():
    print(f"{key}: {value}")
```

حذف جميع العناصر باستخدام clear()

```
person.clear()
print(person)
```

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

Python File Manipulation (التعامل مع الملفات)

التعامل مع الملفات يُعد من الأمور ذات الأهمية الكبيرة في لغات البرمجة، إذ أن تقنيات المعلومات ستفقد معناها إذا لم يكن بالإمكان قراءة البيانات أو حفظها أو استخدامها بشكل دائم.

تشمل الأنواع الشائعة للتعامل مع الملفات: فتح الملفات وإغلاقها، قراءة الملفات وكتابتها، وأخذ نسخ احتياطية من الملفات.

File Manipulation

- **Opening a file**
  - **f.open('file name','access mode')**
  - **Common access modes:**

Access Mode	Description
r	Opens a file only for reading.
w	Opens a file only for writing.
a	Opens a file only for addition.
rb	Opens a file using the binary format only for reading.
wb	Opens a file using the binary format only for writing.
ab	Opens a file using the binary format only for addition.
r+	Opens a file for reading.
w+	Opens a file for writing.
a+	Opens a file for addition.
rb+	Opens a file using the binary format for reading.
wb+	Opens a file using the binary format for writing.
ab+	Opens a file using the binary format for addition.

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

File Manipulation

- Writing data:

```
f = open("name.txt","w")
f.write("libai")
f.close()
```

- Reading data:

```
f = open("name.txt","r")

lines = f.realines()
for line in lines:
    print(line)
```

- Closing a file:

```
f.close()
```

Python Language

كورسات صوت الشيح
القس يا كوبوس بدري

Save Area in File

```
class Rectangle:
    def __init__(self, width, length):
        self.__width = width
        self.__length = length

    def area(self):
        return self.__width * self.__length

    def save_to_file(self, filename):
        # حفظ المساحة في الملف
        file = open(filename, "w")  # فتح الملف في وضع الكتابة
        file.write(f"Rectangle Area: {self.area()}\n")  # كتابة المساحة في الملف
        file.close()  # إغلاق الملف يدويًا
        print(f"Area saved to {filename}")

    def read_from_file(self, filename):
        # قراءة المساحة من الملف
        file = open(filename, "r")  # فتح الملف في وضع القراءة
        content = file.read()  # قراءة المحتوى
        file.close()  # إغلاق الملف يدويًا
        print("Content read from file:")
        print(content)
```

```
# إنشاء كائن من الفئة
rect = Rectangle(5, 10)  # على سبيل المثال

# اسم الملف الذي سيتم الحفظ فيه
filename = "rectangle_area.txt"

# حفظ المساحة في الملف
rect.save_to_file(filename)

# قراءة المساحة من الملف
rect.read_from_file(filename)
```

Python Language

كورسات صوت النسيح
القس باكوبوس بدري

Delete Line , word and Replace (word ,sentence) in File

```
import fileinput

# تعديل كل سطر من الملف مباشرة
for line in fileinput.input('rectangle_area.txt', inplace=True):
    # استبدال النصوص داخل كل سطر
    line = line.replace("Rectangle", "Universe")
    # # طباعة السطر المعدل ليتم كتابته مباشرة إلى الملف
    print(line, end='')

# حذف الكلمة أثناء قراءة الملف وكتابته مباشرة
for line in fileinput.input('rectangle_area.txt', inplace=True):
    line = line.replace("Universe", "")  # حذف الكلمة
    print(line, end='')  # طباعة النص المعدل ليُكتب في الملف

# فتح الملف وتعديل المحتوى مباشرة
for line in fileinput.input('rectangle_area.txt', inplace=True):
    if "Area" not in line:  # تخطي السطر المطلوب حذفه
        print(line, end='')  # طباعة السطور الأخرى فقط
```