ملخص لغة
Python

VARIABLES

نستعمل الـ variables لتخزين البيانات مؤقتًا في ذاكرة الحاسوب

```
price = 10
rating = 4.9
course_name = 'Python for Beginners'
is_published = True
```

في المثال أعلاه:

- price هو integer (عدد صحيح بدون فاصلة عشرية)
- rating هو float (عدد يحتوي على فاصلة عشرية)
- course_name هو string (سلسلة من الحروف)
- is_published هو boolean، ويمكن أن تكون قيمته True أو False.

COMMENTS

نستعمل comments لإضافة ملاحظات داخل الكود. التعليقات الجيدة تشرح السبب والطريقة وليس ما يفعله الكود، لأن ذلك يجب أن يكون واضحًا من الكود نفسه.
استخدم comments لإضافة تذكيرات لنفسك أو للمطورين الآخرين، أو لشرح افتراضاتك والأسباب وراء كتابة الكود بطريقة معينة.

```
# هذا تعليق ولن يتم تنفيذه.
# يمكن أن تكون تعليقاتنا متعددة الأسطر.
```

RECEIVING INPUT

يمكننا استقبال مدخلات من المستخدم باستعمال الدالة input

```
birth_year = int(input('Birth year: '))
```

دالة input() ترجع دائمًا البيانات على شكل string. لذلك نقوم بتحويل النتيجة إلى integer باستعمال الدالة المدمجة int().

STRINGS

يمكننا تعريف strings باستخدام إما علامات اقتباس مفردة ' ' أو مزدوجة " ".

ولتعريف string متعددة الأسطر، نحيط النص بعلامات اقتباس ثلاثية ''' ''' أو """ """.

يمكننا الوصول إلى الحروف داخل string باستعمال الأقواس المربعة [].

```
course = 'Python for Beginners'
course[0]   # يرجع الحرف الأول
course[1]   # يرجع الحرف الثاني
course[-1]  # يرجع الحرف الأخير
course[-2]  # يرجع الحرف قبل الأخير
```

يمكننا أخذ جزء من string باستعمال نفس الصيغة:

```
course[1:5]
```

التعبير أعلاه يُرجع جميع الحروف ابتداءً من الفهرس 1 إلى 5 (باستثناء 5).
النتيجة ستكون: ytho

- إذا حذفنا start index، فالقيمة المفترضة هي 0.
- وإذا حذفنا end index، فالقيمة المفترضة هي طول الـ string.

FORMATTED STRINGS

يمكننا إدراج القيم داخل string بطريقة ديناميكية باستخدام formatted strings:

```
name = 'Amine'
message = f'Hi, my name is {name}'
message.upper()      # لتحويل الحروف إلى حروف كبيرة
message.lower()      # لتحويل الحروف إلى حروف صغيرة
message.title()      # لجعل أول حرف من كل كلمة كبيرًا
message.replace('p', 'q')   # لاستبدال حرف بحرف آخر
message.find('p')   # يرجع موقع أول ظهور للحرف (أو -1 إذا لم يُوجد)
```

للتحقق مما إذا كانت *string* تحتوي على حرف أو سلسلة من الحروف، نستعمل *:in operator*

```
contains = 'Python' in course
```

Arithmetic Operations

```
+     #    الجمع

-     #    الطرح

*     #    الضرب

/     #    القسمة - تُرجع float

//    #    القسمة الصحيحة - تُرجع int

%     #    تُرجع الباقي من القسمة

**    #    الأس - x ** y تعني x مرفوعة للقوة y
```

Augmented assignment operator:

بدل أن نكتب:

```
x = x + 10
```

يمكننا اختصارها إلى:

```
x += 10
```

أولوية العمليات (Operator Precedence)

1. parenthesis :الأقواس
2. exponentiation : الأس
3. multiplication / division: الضرب/القسمة
4. addition / subtraction : الجمع/الطرح

IF STATEMENTS

```
if is_hot:
    print("hot day")
elif is_cold:
    print("cold day")
else:
    print("beautiful day")
```

Logical operators:

```
if has_high_income and has_good_credit:
  print("premium")
if has_high_income or has_good_credit:
  print("standard")
is_day = True
is_night = not is_day
```

COMPARISON OPERATORS

```
a > b       # أكبر من
a >= b      # أكبر من أو يساوي
a < b       # أصغر من
a <= b      # أصغر من أو يساوي
a == b      # يساوي
a != b      # لا يساوي
```

WHILE LOOPS

```
i = 1
while i < 5:
    print(i) i += 1
```

FOR LOOPS

```
for i in range(1, 5):
    print(i)
```

range()

دالة range() تُستخدم لإنشاء تسلسل من الأرقام:

```
range(5)        # ينتج 0, 1, 2, 3, 4
range(1, 5)     # ينتج 1, 2, 3, 4
range(1, 5, 2)  # ينتج 1, 3
```

LISTS

```
numbers = [1, 2, 3, 4, 5]
numbers[0]              # يرجع العنصر الأول
numbers[1]              # يرجع العنصر الثاني
numbers[-1]             # يرجع العنصر الأخير
numbers[-2]             # يرجع العنصر قبل الأخير
numbers.append(6)       # يضيف 6 في نهاية القائمة
numbers.insert(0, 6)    # يضيف 6 في الموضع 0
numbers.remove(6)       # يحذف أول ظهور للعدد 6
numbers.pop()           # يحذف آخر عنصر
numbers.clear()         # يحذف جميع العناصر
numbers.index(8)        # يرجع موضع أول ظهور لـ 8
numbers.sort()          # يرتب العناصر
numbers.reverse()       # يعكس ترتيب العناصر
numbers.copy()          # ينشئ نسخة من القائمة
```

TUPLES

الـ tuples تشبه lists، لكنها تُقرأ فقط (read-only).
نستعملها لتخزين مجموعة من العناصر، لكن بعد تعريف tuple لا يمكننا إضافة عناصر جديدة، أو حذف عناصر، أو تعديل العناصر الموجودة.

يمكننا تفريغ (unpack) العناصر من tuple أو list في متغيرات منفصلة:

```
x, y, z = coordinates
```

DICTIONARIES

نستعمل dictionaries لتخزين أزواج من key/value.

```
customer = {
"name": "John Smith", "age": 30,"is_verified": True
}
```

يمكننا استعمال strings أو numbers كمفاتيح (keys) بشرط أن تكون فريدة،
بينما يمكن أن تكون القيم (values) من أي نوع بيانات.

```
print(customer["name"])  # تُرجع "John Smith"
# محاولة الوصول إلى قيمة غير موجود
print(customer["type"])  # هذا السطر سيُسبب خطأ (KeyError)
# استخدام get لتفادي الخطأ وإرجاع قيمة افتراضية إن لم يوجد المفتاح
print(customer.get("type", "silver"))  # تُرجع "silver"

# تحديث قيمة المفتاح "name"
customer["name"] = "New Name"
print(customer["name"])  # تُرجع "New Name"
```

FUNCTIONS

نستعمل functions لتقسيم الكود إلى أجزاء صغيرة.
هذه الأجزاء تكون أسهل في القراءة والفهم والصيانة.
وفي حال وجود أخطاء (bugs) يكون من الأسهل اكتشافها في جزء صغير بدل البرنامج بالكامل.
كما يمكننا إعادة استخدام هذه الأجزاء في أماكن مختلفة.

```
def greet_user(name):
  print(f"Hi {name}")
greet_user("John")
```

Parameters هي أماكن مخصصة (placeholders) للبيانات التي نمررها إلى الـ function.
أما arguments فهي القيم الفعلية التي نمررها أثناء استدعاء الـ function.
لدينا نوعان من arguments:

- Positional arguments: ترتيبها مهم.
- Keyword arguments: ترتيبها لا يهم، ونسبقها باسم الـ parameter.

```python
# Two positional arguments
greet_user("John", "Smith")
# Keyword arguments
calculate_total(order=50, shipping=5, tax=0.1)
```

يمكن للـ functions أن تُرجع (return) قيمًا.
إذا لم نستعمل جملة return، فسيتم إرجاع None بشكل افتراضي.
None هو كائن (object) يُعبّر عن غياب أي قيمة.

```python
# تعريف دالة لحساب مربع العدد
def square(number):
    return number * number
# استدعاء الدالة وتخزين الناتج
result = square(2)
# طباعة النتيجة
print(result)
```

EXCEPTIONS

الـ exceptions هي أخطاء تتسبب في توقف البرنامج عن العمل (crash).
غالبًا ما تحدث بسبب إدخال غير صحيح أو أخطاء في الكود.
من مسؤولية المبرمج التنبؤ بهذه الأخطاء والتعامل معها بطريقة صحيحة لتجنّب انهيار البرنامج.

```python
try:
  age = int(input('Age: '))
  income = 20000
  risk = income / age
  print(age)
except ValueError:
    print('Not a valid number')
except ZeroDivisionError:
    print('Age cannot be 0')
```

CLASSES

نستعمل classes لتعريف أنواع جديدة (new types) في Python

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def move(self):
        print("move")
```

عندما تكون function جزءًا من class، فإننا نسميها method.

الـ classes تُستخدم لتحديد قوالب (templates) أو مخططات (blueprints) لإنشاء objects.

الـ object هو instance من class.

كل مرة ننشئ فيها instance جديد، فإنه يتبع نفس البنية التي حددناها داخل الـ class.

```
point1 = Point(10, 5) point2 = Point(2, 4)
```

الدالة الخاصة init تُسمّى constructor.
يتم استدعاؤها تلقائيًا عند إنشاء object جديد.
نستعملها لتهيئة (initialize) خصائص object عند إنشائه.

INHERITANCE

الـ inheritance هي تقنية لتفادي تكرار الكود.
نستطيع إنشاء base class تحتوي على الخصائص أو methods المشتركة،
ثم نجعل classes أخرى ترثها (inherit) وتستخدمها مباشرة.

```python
class Mammal:
  def walk(self):
    print("walk")

class Dog(Mammal):
    def bark(self):
    print("bark")

dog = Dog()
dog.walk()  # موروثة من Mammal
dog.bark()  # معرفة داخل Dog
```

MODULES

الـ module هو ملف يحتوي على كود Python.
نستعمل modules لتقسيم البرنامج إلى ملفات متعددة،
وبهذا الشكل يصبح الكود أكثر تنظيمًا وسهل الصيانة.
بدل أن يكون لدينا ملف واحد ضخم يحتوي على آلاف الأسطر، نقسمه إلى modules صغيرة ومنظمة.
هناك طريقتان لاستيراد modules:

- استيراد الـ module بالكامل
- أو استيراد عناصر محددة من داخله

```
# استيراد
#converters module
#بالكامل
import converters
converters.kg_to_lbs(5)
# استيراد
# دالة من
#converters module
from converters import kg_to_lbs
kg_to_lbs(5)
```

PACKAGES

الـ package هو مجلد (directory) يحتوي على ملف باسم __init__.py.
يمكن أن يحتوي على module واحد أو أكثر.
نستعمل packages لتنظيم الكود وتقسيمه إلى أقسام منطقية تسهّل القراءة والصيانة.

```
# استيراد
#sales module
#بالكامل
from ecommerce import sales
sales.calc_shipping()
# استيراد
# دالة من
#sales module
from ecommerce.sales import calc_shipping
calc_shipping()
```

PYTHON STANDARD LIBRARY

تأتي لغة Python مع مكتبة ضخمة من modules الجاهزة
تساعد في تنفيذ مهام شائعة مثل:

- إرسال البريد الإلكتروني
- التعامل مع التاريخ والوقت
- توليد أرقام عشوائية
- إدارة الملفات والمجلدات
- وغيرها الكثير، بدون الحاجة لتثبيت مكتبات خارجية.

RANDOM MODULE

```python
import random

# توليد رقم عشري عشوائي بين 0 و 1
random_float = random.random()

# توليد عدد صحيح عشوائي بين 1 و 6
random_int = random.randint(1, 6)

# اختيار عنصر عشوائي من قائمة
members = ['John', 'Bob', 'Mary']
leader = random.choice(members)
```

Pypi

Python Package Index (pypi.org)
هو دليل يحتوي على حزم Python التي ينشرها المطورون من جميع أنحاء العالم.
نستعمل أداة pip لتثبيت أو إزالة هذه الحزم.

```
pip install openpyxl

pip uninstall openpyxl
```

استفدت؟

لا تنسى المتابعة

ومشاركة المنشور لتعم

الفائدة✨

Amine Mohamed

FULLSTACK DEV