Topic 1: Python Data Types

مقدمة إلى أنواع البيانات الأساسية في بايثون

String (Text) Data Type

تعريف النوع النصي (String)

في بايثون، String هو سلسلة من الأحرف والرموز المحاطة بعلامات الاقتباس.

```
name = "Hello, Python!"
```

خصائص النوع النصي:

- يمكن استخدام علامات الاقتباس المفردة أو المزدوجة
- يمكن التعامل معه كمجموعة من الأحرف
- يستخدم للنصوص والكلمات والجمل

```
print(type(name))  # للتحقق من نوع البيانات
```

Number Data Types

الأنواع الرقمية في بايثون

هناك نوعان رئيسيان من الأرقام:

1. الأرقام الصحيحة (Integer):

```
age = 25
```

2. الأرقام العشرية (Float):

```
height = 1.75
```

خصائص الأنواع الرقمية:

- Integer: الأرقام بدون كسور
- Float: الأرقام مع الكسور العشرية
- يمكن إجراء العمليات الحسابية عليها مباشرة

```
print(type(age))    # للتحقق من نوع العدد الصحيح
print(type(height)) # للتحقق من نوع العدد العشري
```

Boolean Data Type

نوع البيانات المنطقي (Boolean)

يمثل القيم المنطقية صح أو خطأ:

```
is_student = True
is_working = False
```

خصائص النوع المنطقي:

- يحتوي على قيمتين فقط: True أو False
- يستخدم في الشروط والمقارنات
- نتيجة العمليات المنطقية

```
print(type(is_student))  # للتحقق من النوع المنطقي
```

Topic 2: Input and Type Conversion

لماذا نحتاج إلى تحويل أنواع البيانات عند استخدام input

مشكلة الإدخال الأساسية

عند استخدام `input()` في بايثون، يتم إرجاع البيانات دائماً كنص (string):

```
age_input = input("أدخل عمرك: ")
```

لماذا نحتاج التحويل؟

- `input()` يعيد دائماً نص (string)
- لا يمكن إجراء عمليات حسابية على النصوص
- نحتاج لتحويل النص إلى رقم للعمليات الحسابية

كيفية التحويل

1. التحويل إلى رقم صحيح:

```
age = int(age_input)
result = age + 5
```

2. التحويل إلى رقم عشري:

```
height = float(input("أدخل طولك بالمتر: "))
```

نصائح مهمة:

- استخدم `int()` للأرقام الصحيحة
- استخدم `float()` للأرقام العشرية
- تأكد من إدخال قيم صحيحة لتجنب الأخطاء

Topic 3: Conditional Statements (if-elif-else)

كيفية استخدام العبارات الشرطية للمقارنة واتخاذ القرارات

الشرط الأساسي (if)

يستخدم للتحقق من شرط واحد:

```
number1 = 10
number2 = 20

if number1 < number2:
    print("number1 أقل من number2")
```

الشرط الإضافي (elif)

يستخدم للتحقق من شروط متعددة:

```
score = 75

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"

print(f"تقديرك هو: {grade}")
```

الشرط الافتراضي (else)

يستخدم كحالة نهائية إذا لم تتحقق الشروط السابقة:

```
if number1 > number2:
    print("number1 أكبر من number2")
```

```
else:
    print("number1 ليس أكبر من number2")
```

متى نستخدم الشروط؟

- للمقارنة بين القيم
- لاتخاذ قرارات متعددة
- للتحكم في تدفق البرنامج

Topic 4: Rock Paper Scissors with Conditionals and While Loop

كيفية استخدام الشروط والحلقات للعب لعبة الحجر الورقة المقص

إعداد اللعبة | Game Setup

نبدأ بتحديد عدد المحاولات للعبة:

```
number_of_games = 10
```

الحلقة (While Loop) | Game Loop

نستخدم الحلقة للسماح باللعب عدة مرات:

```
while number_of_games > 5:
    # سنبدأ اللعب طالما عدد المحاولات أكبر من 5
    # We'll keep playing as long as game count is above 5
```

اختيار المستخدم | User Choice

نطلب من المستخدم إدخال اختياره:

```
user_input = input("ادخل اختيارك :")
# Prompt the user to enter their choice
```

اختيار الكمبيوتر | Computer Choice

في الكود الأصلي، اختيار الكمبيوتر ثابت:

```
computer_choice = "✊"
# The computer always chooses rock
```

منطق اللعبة | Game Logic

نستخدم الشروط للتحكم في نتيجة اللعبة:

```
if user_input == "👊":
    print("🤝 تعادل")  # Draw
    print("Draw!")
elif user_input == "📄":
    print("you WOONN 🎇🎇")  # Win
    print("You Win!")
elif user_input == "✂️":
    print("you LOSE 😔")  # Lose
    print("You Lose!")
```

تقليل عدد المحاولات | Reduce Game Count

نقلل عدد المحاولات في كل جولة:

```
number_of_games = number_of_games - 1
# Decrease the number of games by 1
```

الشرح الكامل | Full Explanation

لماذا نستخدم الحلقة (While Loop)؟ | Why Use a While Loop?

- تكرار المهام عدد محدد من المرات
- مثالية للألعاب والتفاعلات المتكررة
- تستمر طالما الشرط صحيح

Why Use Conditionals?

- للمقارنة بين القيم
- لاتخاذ قرارات متعددة
- للتحكم في تدفق البرنامج

ملاحظات مهمة | Important Notes

- الكود يستخدم اختيار ثابت للكمبيوتر
- يمكن تطوير اللعبة لتكون أكثر تعقيداً
- الشروط تحدد نتيجة كل جولة

مثال كامل | Full Example

```
number_of_games = 10

while number_of_games > 5:
    user_input = input("ادخل اختيارك :")
    computer_choice = "👊"

    if user_input == "👊":
        print("🤝 تعادل")  # Draw
        print("Draw!")
```

```
    elif user_input == "📄":
        print("you WOONN 🌐🌐")  # Win
        print("You Win!")
    elif user_input == "✂️":
        print("you LOSE 😒")  # Lose
        print("You Lose!")

    number_of_games = number_of_games - 1
```

تحدي للمتعلم | Learner Challenge

- حاول إضافة المزيد من الخيارات للعبة
- فكر في كيفية جعل اختيار الكمبيوتر عشوائياً
- أضف نظام نقاط للعبة

Key Concepts Covered

المفاهيم الرئيسية التي تم تغطيتها في هذه الجلسة:

1. **أنواع البيانات**: فهم النصوص والأرقام والقيم المنطقية
2. **تحويل أنواع البيانات**: التعامل مع الإدخال وتحويل الأنواع
3. **العبارات الشرطية**: استخدام if و elif و else
4. **عوامل المقارنة**: مقارنة القيم واتخاذ القرارات
5. **الحلقات**: إنشاء منطق متكرر للألعاب

Advanced: Common Pitfalls and Best Practices

نصائح متقدمة للتعامل مع أنواع البيانات والشروط

Concept	Common Mistake	Best Practice	Solution
تحويل الأنواع	نسيان تحويل الإدخال	دائماً حول قبل العمليات	استخدم float() أو int()
الشروط	شروط معقدة جداً	اجعل الشروط بسيطة وواضحة	فصل المنطق المعقد
الحلقات	الحلقات اللانهائية	تأكد من وجود طريقة للخروج	استخدم شروط واضحة

تجنب الأخطاء الشائعة في البرمجة وكتابة كود أكثر كفاءة