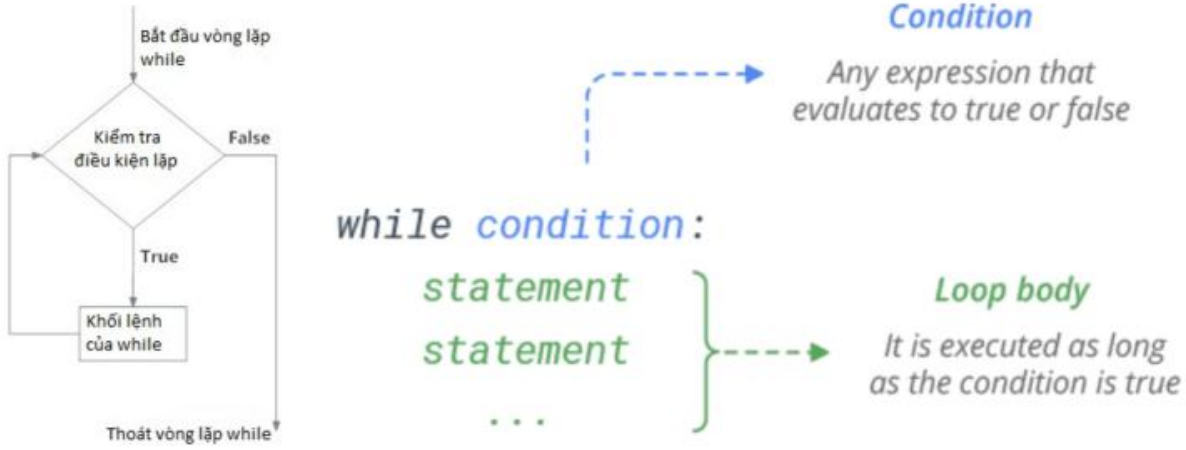
Basic Python - While-Loop

Hoàng-Nguyên Vũ

1. **Mô tả:**

   - Sự ngẫu nhiên là một đặc điểm cơ bản của các hiện tượng trong thực tế. Nó thể hiện qua việc kết quả của một thí nghiệm không thể dự đoán trước được một cách chính xác. **Ví dụ:** Chọn ngẫu nhiên hai số a và b sao cho tổng của a + b bằng 40, câu hỏi đặt ra rằng, chúng ta phải tạo ngẫu nhiên bao nhiêu lần để thỏa mản điều kiện trên ?
   - **Cấu trúc vòng lặp While-Loop:** Vòng lặp while là một cấu trúc điều khiển trong Python cho phép thực thi một khối mã nhiều lần miễn là điều kiện cho trước vẫn còn đúng. Cấu trúc câu lệnh như sau:



2. **Bài tập: về sự ngẫu nhiên - Dùng vòng lặp While**

   - Chọn ngẫu nhiên hai số a và b thuộc từ 1 đến 20 sao cho tổng của a + b bằng 40, câu hỏi đặt ra rằng, chúng ta phải tạo ngẫu nhiên bao nhiêu lần để thỏa mản điều kiện trên ?

```
import random
def random_number_with_condition(total):
    # Set a random value that is the same between devices
    random.seed(0)
    # Your code here

```

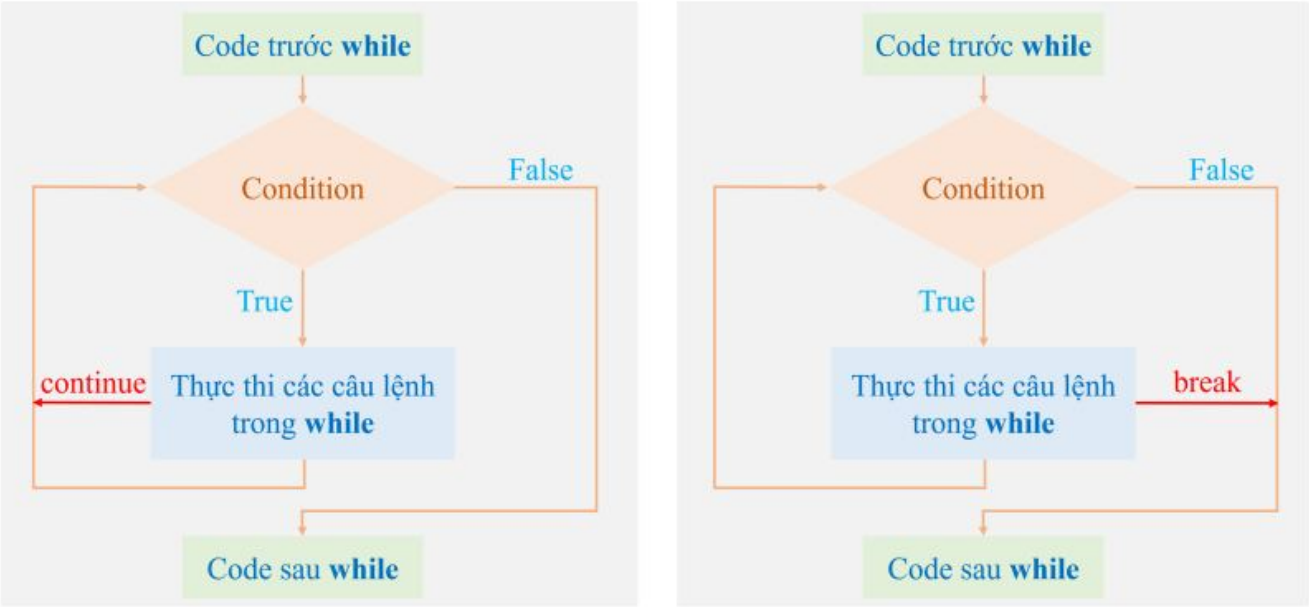
Ví dụ:
- Test case 1: random_number_with_condition(40) → 343 lần
- Test case 2: random_number_with_condition(20) → 32 lần
- Test case 3: random_number_with_condition(35) → 96 lần

Basic Python
While Loop + continue and break keywords

Bảo-Sơn Trần

1. **Mô tả:**

   - Trong Python, câu lệnh **while** được sử dụng để tạo một vòng lặp, trong đó các biểu thức được kiểm tra và nếu điều kiện đúng, khối mã lệnh bên trong vòng lặp sẽ được thực thi. Các lệnh **break** và **continue** được sử dụng để kiểm soát luồng của vòng lặp.



Hình 1: Vòng lặp while kết hợp với từ khóa continue và break.

   - Câu lệnh **break** được sử dụng để thoát khỏi vòng lặp ngay lập tức.
   - Câu lệnh **continue** được sử dụng để bỏ qua phần còn lại của vòng lặp và chuyển đến lần lặp tiếp theo.

2. **Bài tập:** Cài đặt hàm find_divisible_number(a) tìm số nguyên dương nhỏ nhất lớn hơn 100 và chia hết cho số nguyên dương a

```python
def find_divisible_number(a):
    """
    Find the smallest integer that is greater than 100 and divisible
    by the integer a
    """

    #TODO: Your code here
```

Ví dụ:
- Test case 1: find_divisible_number(5) → 105
- Test case 2: find_divisible_number(17) → 102

Basic Python - While-Loop Exercise

Bảo-Sơn Trần

1. **Mô tả:**

   - Phương pháp Newton (Newton's Method), còn được gọi là phương pháp Newton-Raphson, là một phương pháp số học để tìm gần đúng của các nghiệm của một hàm số thực. Cụ thể, nó thường được sử dụng để tìm gần đúng của các nghiệm của phương trình f(x) = 0.
   - Ngoài ứng dụng trong tìm nghiệm của một hàm số, phương pháp Newton còn có ứng dụng trong máy học (Machine learning) trong việc tìm nghiệm của đạo hàm của hàm loss. Tuy nhiên đây là phương pháp không phổ biến bằng thuật toán gradient descent.
   - Ở bài này, chúng ta sẽ dùng phương pháp Newton để tính căn bậc hai cho một số dương a. Chúng ta thực hiện các bước sau:

     (a) Khởi tạo giá trị $x_0 = a$, $n = 0$ và cho trước giá trị ε (thực ra x_0 có thể nhận bất kỳ giá trị dương nào). Tiếp đó, ta sẽ đi xây dựng hàm $f(x) = x^2 - a$. Ở đây, ta xem x_n (ở bước hiện tại đang là x_0) chính là lời giải cho bài toán tính căn bậc hai của a.

     (b) Cải thiện xấp xỉ x_n bằng xấp xỉ x_{n+1} theo công thức tổng quát như sau: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

     (c) So sánh xấp xỉ x_{n+1} và x_n. Nếu $|x_{n+1} - x_n| < \varepsilon$, ta sẽ dừng việc cải thiện xấp xỉ, và trả về kết quả căn bậc hai của a là x_{n+1}. Ngược lại thực hiện tiếp bước (b) với $n = n + 1$.

2. **Bài tập:** Cài đặt hàm find_squared_root(a) tìm căn bậc hai cho một số a bất kì với $\varepsilon = 0.001$.

```
def find_squared_root(a):
    """Find the squared root of number a"""
    EPSILON = 0.001

    #TODO: Your code here

```

Ví dụ:
- Test case 1: find_squared_root(2) → 1.4142135623746899
- Test case 2: find_squared_root(3) → 1.7320508100147276