Introduction to Python

While, Nested Loops, List of Tuples

Indefinite Loop

A **for** loop, we discussed earlier is an example of a **definite loop**, the number of iterations can be specified ahead of time by the programmer.

In some cases, however, the number of iterations can be unknown. This type of loop is called an **indefinite loop.** For example, a user is asked to enter a sequence of positive numbers. The user can enter 0 to finish with their inputs. The number of inputs the user enter is not known in advance.

In this section, we explore the use of the **while** loop to describe conditional iteration: iteration that repeats as long as a condition is true.

Between the two loops, the for loop is more common.

While Loop

The **while** loop is used to describe conditional iteration: iteration that repeats as long as a condition is true.

```
while <condition>:
      block
```

The loop **r**epeatedly executes its block of code as long as the condition is true. The first time the condition is false, the while loop terminates.

At least one statement in the block of the loop must update a variable that affects the value of the condition. Otherwise, the loop will continue forever, an error known as an **infinite loop**.

Definite vs Indefinite Loop

For loops are commonly used for definite loops: loops where the number of iterations is known in advance. For example, count the number of prime numbers between 1 and 1000. Loop 1000 times from 1 to 1000.

While loops are used for indefinite loops: loops where the number of iterations is unknown in advance. For example, generate and print out random numbers from 1 to 10. Stop when there are a total of 5 primes.

Sample run(9 iterations): 4 6 5 5 7 8 10 2 2

Sample run(5 iterations): 5 7 2 5 2

We can't predict the number of iterations required before the loop terminates.

Examples of Definite loops

Definite loops are loops whose number of iterations can be calculated in advance.

1) Print ”hello” 10 times.

```
for i in range(10):
      print(“hello”)
```

This loop always run 10 times.
It is a definite loop.

2) Count the number of primes up to n.

```
def count_primes(n):
      count = 0
      for i in range(1, n+1):
            if is_prime(i):
                  count = count + 1
      return count
```

This loop in the function always run n times. It is a definite loop.

Example of An Indefinite Loop

Indefinite loops are loops whose number of iterations is unknown.

Suppose we want to write a function that returns the largest prime less than or equal to n. Let's do some examples:

n = 10

10	not prime, next!
9	not prime, next!
8	not prime, next!
7	is prime, done!

Checked 4 numbers.

n = 14

14	not prime, next!
13	is prime, done!

Checked 2 numbers.

Note: For a general n, the number of iterations required is unknown.

Example of An Indefinite Loop

Suppose we want to write a function that returns the largest prime less than or equal to n.

In the previous slide, we repeatedly decrease by 1 as long as(while) the current value is not prime.

```
def largest_prime(n):
    i = n              # start with n
    while not is_prime(i):    # while i is not prime
        i = i - 1             # decrease by 1 and repeat
    return i             # while loop termintes if i is prime
```

While Loop

Here's another example of a loop where the number of iterations is unknown in advance. Suppose a program computes a sum of a set of inputs given by the user. The user presses enter to quit. We don't know when the user finishes entering the inputs. Instead of a for loop, a while loop is more appropriate.

```
s = 0.0
data = input("Enter a number or press ENTER to quit: ")
while data != "":
    data = float(data)
    s += data
    data = input("Enter a number or press ENTER to quit: ")
print("The sum is", s)
```

Sample Run

```
s = 0.0
data = input("Enter a number or press ENTER to quit: ")
while data != "":
    data = float(data)
    s += data
    data = input("Enter a number or press ENTER to quit: ")
print("The sum is", s)
```

Sample Run:

Enter a number or just enter to quit: 3

Enter a number or just enter to quit: 4

Enter a number or just enter to quit: 5

Enter a number or just enter to quit:

The sum is 12.0

Random Numbers

In some situation, we like to be able to simulate randomness. For example, we might toss a coin or roll a die.

The Python's random module contains many functions to do this. The function **randrange()** is easy to use since it is similar to the **range()** function we used in for loops. We must first import the random module to access its code. This is done using the statement "import random".

randrange(start, stop, step) generates a random integer beginning with start(including) and ending with stop(not including) with step.

```
import random
for i in range(10):
    num = random.randrange(1, 5)
    print(num, end=" ")
```

Output:

3 3 3 2 1 3 2 2 3 4

Generate random sequence

Write a segment of code that prints out a sequence of random numbers from 1 to 100. Stop once exactly 5 prime numbers have been printed.

Note that, we don't know how many iterations we need to get 5 prime numbers. This is an indefinite loop. It's better to use a while loop.

```
import random
count = 0
while count < 5:
    num = random.randrange(1, 101)
    print(num, end=" ")
    if is_prime(num):
        count += 1
```

Output:

3 26 22 5 68 19 42 58 23

AP Exam: Repeat Until

The AP exam does not have a while loop. It has something equivalent called a "Repeat Until" loop.

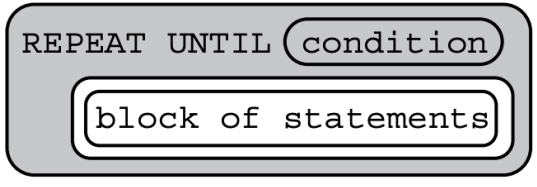
In Python, a while loop repeats as long as the condition is true.

On the AP exam, a repeat until loop repeats until the condition is true.

Text:

```
REPEAT UNTIL(condition)
{
 <block of statements>
}
```

Block:



The code in `block of statements` is repeated until the Boolean expression `condition` evaluates to `true`.

AP Exam: Repeat Until

A Python while loop can be translated to a repeat until loop and vice versa:

The following two loops are equivalent:

Python

```
while condition:
    block of statements
```

This loop repeats as long as condition is True and terminates when condition is False.

AP Exam

```
REPEAT UNTIL(not condition):
    block of statements
```

This loop repeats until "not condition" is true, that is, it repeats until condition is False. In other words, it also terminates when condition is False.

AP Exam: Repeat Until

Here is an example.

The following two loops are equivalent:

Python

```
count = 0
while count < 5:
    count = count + 2
    print(count, end=" ")
```

This loop repeats as long as count < 5 and terminates when count >= 5

AP Exam

```
count ← 0
REPEAT UNTIL(count >= 5):
    count ← count + 2
    DISPLAY(count)
```

This loop repeats until count >= 5, that is, it also repeats as long as count < 5 and termintes when count >= 5

Output:
2 4 6

Nested Loops

A *nested loop* is a loop inside of another loop.

```
for i in range(1, 4):
      print(i, end=':')
      for j in range(1, 5):
            print(j, end=' ')
      print()
```

```
1:1 2 3 4
2:1 2 3 4
3:1 2 3 4
```

Nested Loops Example 1

```
for i in range(1, 6):
    for j in range(1, i+1):
        print(j, end=' ')
    print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Sum of 2D lists

A 2D list is a list of lists. It can represent data from an excel sheet or pixels in an image(more on this in a later lecture).

```
list2d = [[1, 2, 5],
          [3, -1, 6],
          [10, 1, 0]]
print(list2d[0])    # [1, 2, 5]
print(list2d[0][1]) # 2
print(list2d[2][0]) # 10
```

Equivalently:

```
list2d = [[1, 2, 5],[3, -1, 6],[10, 1, 0]]
```

Sum of 2D lists

Nested loops is often used to traverse 2-dimensional data.

```
list2d = [[1, 2, 5],
          [3, -1, 6],
          [10, 1, 0]]
sum = 0
for row in list2d:
     for num in row:
          sum += num
print(sum)
```

Equivalently:

```
list2d = [[1, 2, 5],[3, -1, 6],[10, 1, 0]]
```

Output:

27

Sum of 2D lists(Using Indices)

A 2D list is a list of lists.

```
list2d = [[1, 2, 5],
          [3, -1, 6],
          [10, 1, 0]]
sum = 0
for i in range(len(list2d)):
     for j in range(len(list2d[i])):
          sum += list2d[i][j]
print(sum)
```

Equivalently:

```
list2d = [[1, 2, 5],[3, -1, 6],[10, 1, 0]]
```

Output:

27

Tuples as Records

Often we have data that contains several pieces of information rather than just one. For example, a student data may contain both their name and gpa. A variable containing information about a car may contain its make, model and year.

Tuples is a list-like datatype that is used as **records,** containing a collection of fields.

```
student = (“Mike Smith”, 3.5)  # (name, gpa) tuple
car = (“Tesla”, “X”, 2017)
```

They can also be defined without any brackets at all:

```
student = “Mike Smith”, 3.5
car = “Tesla”, “X”, 2017
```

Tuples

Like the lists, tuples have a length and individual elements can be extracted using square-bracket indexing. Slicing is also supported. Unlike lists, however, tuples are immutable: it cannot be modified!

```
student = (“Mike Smith”, 3.5)
print(len(student))          # 2
print(student[0])            # Mike Smith
print(student[1])            # 3.5
student[0] = “Mike Smith”    # error! tuples are immutable
```

Rather than accessing tuple data using indices like above, we usually will use tuples unpacking to unpack a tuple into meaningful variables:

```
name, gpa = (“Mike Smith”, 3.5)
make, model, year = “Tesla”, “X”, 2017
```

List of Tuples

We can form a list of tuples to store a collection of items, each of which consists of more than one pieces of information. For example, a list of students where each student object contains a name and a gpa.

Here's an example of a for loop that uses tuple unpacking.

```
students = [("Mike", 3.2), ("Sarah", 3.6), ("Jack", 2.8)]
sum = 0
for name, gpa in students:  # tuple unpacking
      sum += gpa
ave = sum / len(students)
```

Note: You will be asked to write a program and submit to the College Board as part of your portfolio. This is called the Create Task. One of the requirements is to have a list of data. A list of tuples is a good way to fulfill this requirement.

Lab 1

In this lab, we will write a simple guessing game.

The computer randomly generate a number in some given range. On each pass through the loop, the user enters a number to attempt to guess the number selected by the computer.

The program responds by saying "You've got it," "Too large, try again," or "Too small, try again." When the user finally guesses the correct number, the program congratulates him and tells him the total number of guesses.

There is a template repl for this lab at here:

https://repl.it/@LongNguyen18/GuessingGameLab

Let's look at a sample run. What's the strategy for guessing the number?

Guessing Game

```
Enter the smaller number: 10
Enter the larger number: 5
Error! The small number must be <= the larger number. Try again!
Enter the smaller number: 1
Enter the larger number: 100
Enter your guess: 50
Too small!
Enter your guess: 75
Too large!
Enter your guess: 62
Too small!
Enter your guess: 68
Too large!
Enter your guess: 65
You've got it in 5 tries!
```

There is a template repl for this lab at here:

https://repl.it/@LongNguyen18/GuessingGameLab

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.
2) Lambert, Kenneth, Fundamentals of Python: First Programs, Cengage Learning, 2017.