Introduction to Python

For Loops and Strings

Topics

1) For Loops
2) Strings

Iteration

Iteration is a repeating portion of an algorithm. Iteration repeats a specified number of times or until a given condition is met.

For example, if we want to print a message 10 times:

```python
for i in range(5):
    print("hi")
```

hi
hi
hi
hi
hi

Iteration

Iteration loops are frequently referred to as **for** loops because for is the keyword that is used to introduce them in nearly all programming languages, including Python.

Python's *for* loop iterates over items of a sequence(e.g. a list of numbers or a string(sequence of characters)) and process them with some code.

```python
for x in sequence:
    block
```

For Loops

Python's *for* loop iterates over items of a sequence(e.g. a list of numbers or a string(sequence of characters)) and process them with some code.

```
for x in sequence:
    block
```

This is a list. It is an example of a sequence.
More on lists in a later lecture.

```
for x in [2,3,5,7]:
    print(x, end=" ")        # ends each print with a space
                             # print all on same line
```

2 3 5 7

For Each Loops

```
for x in [2,3,5,7]:
    print(x)

2

3

5

7

for x in "hello":
    print(x)

h

e

l

l

o
```

Iterate through each number in the list.

"for each x in" list.

Iterate through each character in the string.

range(stop)

A simple use of a *for* loop runs some code a specified number of times using the *range()* function.

range(stop): returns sequence of numbers from 0 (default) up to but not including stop. Increment by 1 (default).

```
for i in range(5):
    print(i, end=" ")
```

0 1 2 3 4

Think of range(5) as generating this list: [0, 1, 2, 3, 4].

```
for i in range(10):
    print("hi", end=" ")
```

hi hi hi hi hi hi hi hi hi hi

range(start, stop)

range(start, stop): from start up to but not including stop. Increment by 1 (default).

```python
for i in range(2, 8):
    print(i, end=' ')
```
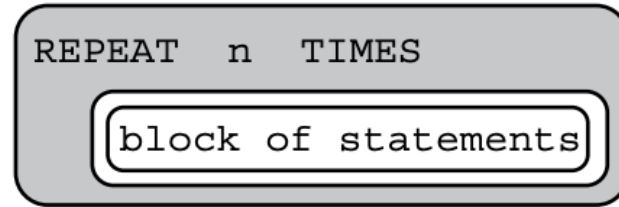
2 3 4 5 6 7

AP Exam

To perform a task n times, the AP exam use the following syntax.

```
Text:
REPEAT n TIMES
{
 <block of statements>
}
Block:
```

```
REPEAT   n   TIMES
    block of statements
```

Note: The above is equivalent to the following code in Python:

```python
for i in range(n):
        # block of statements

        ...
```

Definite Iteration

The for loop is an example of a **definite iteration**. We can determine ahead of time the number of times the loop repeats. Later, we will talk about **indefinite iteration**, a loop where we cannot predict the number of times a loop repeats.

```python
for i in range(5):
    print("*", end="")
```

*****

The loop above prints five *'s. We can determine this ahead of time from the for loop statement.

Summing and Counting

There are two common tasks that uses for loops.

1) Summing
2) Counting

Summing Values

Write a segment of code that solve the problem

$$1 + 2 + 3 + \ldots + 98 + 99 + 100.$$

We need a variable that accumulate the sum at each iteration of the loop. This variable should be initialized to 0.

```python
sum = 0
for i in range(1, 101):
    sum += i
```

Writing a function to sum

Now write a function that accepts a non-negative integer parameter n and returns the sum of integers from 1 to n(including).

```python
def sum(n):
    sum = 0
    for i in range(1, n+1):
        sum += i
    return sum


print(sum(5))    # 1+2+3+4+5=15
a = sum(100)     # a = 5050
print(a)         # 5050 is printed on console
```

Conditional Summing

Write a segment of code that compute the sum of all numbers from 1 to 100 that are multiples of 3. Loop but only sum if a certain condition is true.

```python
sum = 0
for i in range(1, 101):
    if i % 3 == 0:
        sum += i
```

Conditional Summing Example

Write a segment of code that compute the sum of all numbers from 1 to 100. However:

1) if a number is a multiple of 3, double it before adding,

2) if a number is a multiple of 5, triple it before adding,

3) If a number is a multiple of both, quadruple it before adding.

4) otherwise, just add the number.

Conditional Summing Solution?

Is the following a correct solution?

```python
sum = 0
for i in range(1, 101):
    if i % 3 == 0:
        sum += 2 * i
    elif i % 5 == 0:
        sum += 3 * i
    elif i % 3 == 0 and i % 5 == 0:
        sum += 4 * i
    else:
        sum += i
```

No! Why not?

Conditional Summing Solution

The following is correct.

```
sum = 0
for i in range(1, 101):
        if i % 3 == 0 and i % 5 == 0:
                sum += 4 * i
        elif i % 3 == 0:
                sum += 2 * i
        elif i % 5 == 0 :
                sum += 3 * i
        else:
                sum += i
```

Counting

Write a function that accepts an integer parameter n and returns the number of factors of n.

```python
def count_factors(n):
    count = 0
    for i in range(1, n+1):
        if n % i == 0:        # i is a factor of n
            count += 1
    return count

print(count_factors(10))  # 4 (factors of 10 = {1,2,5,10})
print(count_factors(7))   # 2 (factors of 7 = {1,7})
```

String Indexing

Python allows you to retrieve individual members of a string by specifying the *index* of that member, which is the integer that uniquely identifies that member's position in the string. The built-in len() function returns the number of characters in a string.

```python
message = "hello"
length = len(message)
print(length)          # 5
print(message[0])      # h
print(message[1])      # e
print(message[4])      # o
print(message[5])      # error! out of range!
```

Note: On the AP exam, The first index of the first character of a string is 1 not 0.

String Indexing

Strings are **immutable**: once it is created, it cannot be changed!

```python
message = "hello"

message[0] = "H"          # ERROR! A string is immutable!
```

Negative indices can be used to access characters of a string. The last character is at index **-1**, the second to last at index **-2**, etc…

```python
print(message[-1])        # o

print(message[-2])        # l
```

Looping Through Each Character

Since each character of a string has index, we can use a loop to traverse a string.

```python
message = "hello"
for i in range(len(message)):
    print(message[i])
```

Output:

h

e

l

l

o

Slicing

We can also "slice" a string, specifying a start-index and stop-index, and return a subsequence of the items contained within the slice.

Slicing is a very important indexing scheme that we will see many times in other data structures(lists, tuples, strings, Numpy's arrays, Panda's data frames, etc..). Slicing can be done using the syntax:

`some_string[start:stop:step]`

where

start: index of beginning of the slice(included), default is 0

stop: index of the end of the slice(excluded), default is length of string

step: increment size at each step, default is 1.

Slicing

```python
language = "python"
print(language[0:4])   # pyth
                       # 0 up to but not including index 4
print(language[:4])    # pyth, default start index at 0
print(language[4:])    # on, default end index is length of string

print(language[:])     # python, 0 to end of string
print(language[0:5:2]) # pto, step size of 2
print(language[::-1])  # negative step size traverses backwards
                       # nohtyp
```

Functions on Strings

Functions we discussed so far are isolated, independent entities. Sometimes functions are associated with some object and operates on the data of that object. In this context, functions are called **methods**.

Strings is an example of a type of objects which contains methods. These methods can be accessed through the **dot notation** applied to a string variable or literal.

find(value)	returns the lowest index of a substring value in a string. If substring is not found, returns -1.
upper() and lower()	returns a copy of the string capitalizing(or lower casing) all characters in the string

String Methods

```python
s = "Hi, Mike!"

index = s.find("Hi")
print(index)                 # 0, first letter's index is 0.
print(s.find(" "))      # 3
print(s.find("Mike"))    # 4


index2 = s.find("mike")  # -1, not found
b = "python"
print(b.upper())             # PYTHON
print("JAVA".lower())    # java
```

String Methods

Note that upper(), lower() do not modify the original string but rather returns a new copy of the string.

```
s = "HI MIKE"
s.lower()                    # returned value "hi mike" is lost
print(s)                     # HI MIKE (s is unchanged)


s = s.lower()  # store the modified, returned string back in s
print(s)       # hi mike
```

f-Strings

f-Strings is the new way to format strings in Python. (v 3.6)

Also called "formatted string literals," f-strings are string literals that have an f at the beginning and curly braces containing expressions that will be replaced with their values.

```python
name = "Mike"
gpa = 3.2
f_str = f"I am {name} with a {gpa} gpa."
print(f_str)
print("I am " + name + " with a " + str(gpa) " gpa.")
Output:
I am Mike with a 3.2 gpa.
I am Mike with a 3.2 gpa.
```

f-Strings

An f-string is special because it permits us to write Python code *within* a string; any expression within curly brackets, {}, will be executed as Python code, and the resulting value will be converted to a string and inserted into the f-string at that position.

```python
grade1 = 1.5
grade2 = 2.5
ave = f"average is {(grade1+grade2)/2}"
print(ave)      # average is 2.0
```

This is equivalent but it is preferable to use an f-string.

```python
average = "average is " + str((grade1+grade2)/2)
```

AP Exam

There will be questions on the AP exam which requires students to manipulate strings. The exam will provide an API(application programming interface) for string manipulation. A sample API below was given in previous exams.

Procedure Call	Explanation
concat(str1, str2)	Returns a single string consisting of str1 followed by str2. For example, concat("key", "board") returns "keyboard".
reverse(str)	Returns the reverse of the string str. For example, reverse("abcd") returns "dcba".
substring(str, start, length)	Returns a substring of consecutive characters from str, starting with the character at position start and containing length characters. The first character of str is located at position 1. For example, substring("delivery", 3, 4) returns "live".
substring(str, start, end)	Returns a substring of consecutive characters of str starting with the character at position start and ending with the character at position end. The first character of str is considered position 1. For example, substring("delivery", 3, 6) returns "live".
len(str)	Returns the number of characters in str. For example, len("pizza") returns 5.

The substring function may take on different arguments on the exam. Read the API carefully as this may vary year to year.

For Loop in Movies and TV-Shows

Movies:

Groundhog Day(1993); Bill Murray.

Looper(2010); Bruce Willis and Joseph Gordon-Levitt, Emily Blunt.

Edge of Tomorrow(2014); Tom Cruise, Emily Blunt.

Happy Death Day(2017).

TV-Show:

Russian Doll(Netflix, Emmy-Nominated)

Lab 1

Create a new repl on repl.it.

Write **a for loop** to do each of the following:

1) Print out "Hello!" 10 times, each on a different line.
2) Alternate between printing "Hello" and "Hi" for a total of 20 times, each on a separate line. Use only one for loop. (Hint: Use a conditional)
3) Print 1 4 9 16 25 … 100
4) Print 10 8 6 4 2 0 -2
5) Compute the sum: $1^2+2^2+3^2+4^2+…+19^2+20^2$

Lab 2: Counting Primes

Create a new repl.

1) Rewrite the function count_factors as explained in a previous slide.

2) A number n is prime if its only factors are 1 and n. Write the function is_prime which accepts an integer n and returns whether it is prime. Note that 1 is not prime. You must call the function count_factors in your implementation of is_prime.

is_prime(13) returns True

is_prime(1245) returns False

3) Write the function num_primes which accepts an integer n and and returns the number of primes up to and including n. You must call the function is_prime in your implementation.

num_prime(11) returns 5 since 2, 3, 5, 7, 11 are the 5 prime numbers less than or equal to 11.

Call the three above functions with different inputs and make sure that your functions work as expected.

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.