시몬 베드로가 대답하여 이르되 주는 그리스도시요 살아 계신 하나님의 아들이시니이다 예수께서 대답하여 이르시되 바요나 시몬아 네가 복이 있도다 이를 네게 알게 한 이는 혈육이 아니요 하늘에 계신 내 아버지시니라 또 내가 네게 이르노니 너는 베드로라 내가 이 반석 위에 내 교회를 세우리니 음부의 권세가 이기지 못하리라 내가 천국 열쇠를 네게 주리니 네가 땅에서 무엇이든지 매면 하늘에서도 매일 것이요 네가 땅에서 무엇이든지 풀면 하늘에서도 풀리리라 하시고 (마16:16-19)

Python for Machine Learning

Lecture Notes by idebtor@gmail.com, Handong Global University

NOTE: The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to `idebtor@gmail.com` Your assistances and comments will be appreciated.

Revisit Data Types

- Primitive data types: `int, float, complex, bool, str`
- Collection data types: `list, tuple, dict, set`

Example:

In [ ]:
```
fruits = ["apple", "banana", "kiwi", "orange"]    #list - mutable, ordered
counts = (5, 7, 3)                                #tuple - immutable, ordered
stocks = {'apple': 5 , 'banana': 7, 'orange': 3}  #dictionary - key:value pair, not or
cities = {'daegue', 'pusan', 'jeju' , 'pohang'}   #set - no duplicate, not ordered

print(fruits)
print(counts)
print(stocks)
print(cities)
```

In [ ]:
```

```

Lesson - Iteration

학습 목표

- 반복문에 대해 이해한다.
- 반복문을 적절히 사용하여 문제를 해결할 수 있다.

학습 내용

1. break 명령어 사용하기
2. Nested loops 이해하기
3. for문 & range()
4. Enumerate()

Iteration using for loop

Iteration means executing the same block of code over and over, potentially many times. A programming structure that implements iteration is called a loop. Often the program needs to repeat some code many times. That's where the loops come in handy. There are `for` and `while` loop operators in Python, in this lesson we cover for.

The `for` loop iterates over any sequence that is either a list, a tuple, a dictionary, a set, or a string. With the `for` loop we can execute a set of statements, once for each item in a list, tuple, set etc.

For instance, you may print each fruit in a fruit list:
Expected Output:

```
apple
banana
kiwi
orange
```

In [1]:

```
fruits = ["apple", "banana", "kiwi", "orange"]    #list - mutable, ordered

None
```

The for loop does not require an indexing variable to set beforehand.

Even strings are iterable objects, they contain a sequence of characters. You may loop through the letters in the word `apple`:
Expected Output:

```
a
p
p
l
e
```

In [ ]:

```
None
```

The break statement

With the break statement we can stop the loop before it has looped through all the items:

For example, While print each fruit in the fruit list, exit the loop when it encounters "kiwi" without printing it.

Expected Output:

```
apple
banana
```

In [ ]:

```
fruits = ["apple", "banana", "kiwi", "orange"]    #list - mutable, ordered
```

```
None
```

Nested loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

Expected Output:

```
red apple
red banana
red kiwi
big apple
big banana
big kiwi
tasty apple
tasty banana
tasty kiwi
```

In [ ]:

```
looks = ["red", "big", "tasty"]
fruits = ["apple", "banana", "kiwi"]    #list - mutable, ordered

None
```

Example 1:

Print each fruit in a list and add "my favorite" if it encounters "kiwi".as shown below:

```
fruits = ["apple", "banana", "kiwi", "orange"]
```

Expected output:

```
apple
banana
kiwi - my favorite
orange
```

In [ ]:

```
fruits = ["apple", "banana", "kiwi", "orange"]    #list - mutable, ordered

None
```

Example 2:

Using `for` loop, print `hello` in one line.

Expected output:

```
h e l l o
```

Hint: By default, the function `print()` prints all its arguments separating them by a space and the puts a newline symbol after it. This behavior can be changed using keyword arguments `sep` (separator) and `end` .

In [ ]:
```
# print each letter using for loop in a row
None
```

Example 3:

Using `for` loop, print `hello` in one line wihtout spaces between chars.

Expected output:

```
hello
```

In [ ]:
```
# print each letter using for loop in a row without spaces
None
```

Example 4:

Print `12:3` when `hour = 12` and `min = 3` are given.

Expected output:

```
12:3
```

In [ ]:
```
hour = 12
min = 3
print(None)
```

Example 5:

Using print format - Print the following hours and mins.

Expected output:

```
12:03
02:05
01:45
23:32
```

step.1 `12:3` - it may use 'sep=''' separator option

In [ ]:
```
print(None)
```

step.2 `12: 3`

In [ ]:
```
print(None)
```

step.3 `12:03`

In [ ]:
```
print(None)
```

step.4

```
12:03
02:05
01:45
23:32
```

In [ ]:
```
hours = [12, 2,  1, 23]
mins  = [ 3, 5, 45, 32]
None
```

Iteration with for loop & range()

Another use case for a for-loop is to iterate some integer variable in increasing or decreasing order. Such a sequence of integer can be created using the built-in function `range()`. The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Syntax

```
class range(stop)
    - returns a sequence of numbers starting from 0 to stop - 1
class range(start, stop[, step])
    - returns the range of integers end at stop - 1.
```

Parameters

- `start` - integer starting from which the sequence is returned
- `stop` - integer before which the sequence of integers is to be returned.
- `step` (Optional) - integer value which determines the step in the sequence

The function `range(stop)` means that `start` is implicitly set to `0`:

Example 1:

Generate numbers from 0 to 5 and print them as shown below:

```
0, 1, 2, 3, 4, 5,
```

- Check what `range(6)` returns.

In [ ]:
```
iter = range(6)
for i in range(6):
    print(i, end=', ')
```

Using `start` and `end` parameters in `range()`

- Function `range(start, stop)` generates a sequence with numbers `start, start + 1, ..., start - 1`.

- **The last number in `stop` is not included**.

Example 2:

Print the table of n cubed where n is 1 to 10 as shown below:

Expected output:

```
1 cubed is 1
2 cubed is 8
3 cubed is 27
4 cubed is 64
5 cubed is 125
6 cubed is 216
7 cubed is 343
8 cubed is 512
9 cubed is 729
10 cubed is 1000
```

In [ ]:
```
for i in None:
    None
```

Example 3:

Given an integer n = 10, sum number from 0 to n.

Expected output:

```
Sum from 1 to 10 is 55
```

In [ ]:
```
n = 10
sum = 0
for i in None:
    None

print('Sum from 1 to {} is {}'.format(n, sum))
```

Example 4:

Count down a number starting n and print `blast off`. Complete the code to produce the following output when N = 5.

Expected output:

```
5
4
3
2
1
blast off
```

Using a negative step size `-1` in `range()`

In [ ]:
```
N = int(input("Enter N: "))
for i in None:
    print(None)
print("blast off")
```

Using `range(N)`

In [ ]:
```
N = int(input("Enter N: "))
for i in None:
    print(None)

print("blast off")
```

Enumerate()

Python gives you the luxury of iterating directly over the values of the list (or iterables) which is most of the time what you need. However, there are times when you actually need the index of the item as well.

Syntax:

```
enumerate(iterable, start=0)
```

Parameters:

- `iterable` : any object that supports iteration
- `start` : the index value from which the counter is to be started, by default it is 0

Example 1:

Print a list in three different ways.

Given list of car names, `cars = ['kia', 'audi', 'bmw', 'genesis']` , list car names with sequential numbers.

Expected output(Case 1):

```
kia
audi
bmw
genesis
```

In [ ]:
```
cars = ['kia', 'audi', 'bmw', 'genesis']
for car in None:
    print(car)
```

Expected output(Case 2):

```
(1, 'kia')
(2, 'audi')
(3, 'bmw')
(4, 'genesis')
```

In [ ]:
```
cars = ['kia', 'audi', 'bmw', 'genesis']
for car in None:
    print(car)
```

Expected output(Case 3):

```
1 kia
2 audi
3 bmw
4 genesis
```

In [ ]:
```
cars = ['kia', 'audi', 'bmw', 'genesis']
for i, car in None:
    print(None, None)
```

Example 2:

Print a list with its index

Given list of names, `friends = ['Jane', 'Peter', 'John', 'Lee']` , modify the code cell such that it prouce the list with sequential numbers.

Expected output:

```
1: Jane
2: Peter
3: John
4: Lee
```

Solution 1: Using an index but without using enumerate()

In [ ]:
```
for i in None:
    print(i + 1, None, None)
```

Solution 2: with using enumerate()

In [ ]:
```
for i, friend in None:
    print(i, None, None)
```

Excercise

Count to N

Get an integer N from the user, print all the numbers from 1 to N twice as shown below.

Expected output:

```
1
2
3
4
```

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

In [ ]:

```
N = int(input("Enter N: "))
None
```

Print even numbers twice

Given an integer N, print all even numbers from 0 to N in ascending order and descending order

Expected Output:

```
Enter N: 5
0 2 4
4 2 0

Enter N: 10
0 2 4 6 8 10
10 8 6 4 2 0

Enter N: 11
0 2 4 6 8 10
10 8 6 4 2 0
```

In [ ]:

```
N = int(input("Enter N: "))
None
print()

None
```

Print every other character in a string

Given a string, print every other character. You may need to use a for-loop, if, len(), and range().

Expected Output:

```
Enter a string: hello
h l o

Enter a string: Hello World
H l o W r d

Enter a string: Hello World!
H l o W r d
```

In [ ]:

```
str = input("Enter a string: ")
None
```

Ladder

For given integer n ≤ 9 print a ladder of n steps. The k-th step consists of the integers from 1 to k without spaces between them. To do that, you can use the sep and end arguments for the function print().

Expected Output:

```
Enter N: 5
1
12
123
1234
12345

Enter N: 3
1
12
123
```

In [ ]:
```
N = int(input('Enter N: '))
None
```

Stair

For given positive integer N ≤ 9 print a stair of N steps. The k-th step consists of k of 1's without spaces between them. To do that, you can use the sep and end arguments for the function print().

Expected Output:

```
Enter N: 5
1
11
111
1111
11111

Enter N: 4
1
11
111
1111
```

In [ ]:
```

```

Sum all digits for any number of digits

For example, print `10` if `316` , `13` if `256` , `26` if `5678` .

Expected Outputs:

```
316
10

256
```

```
13

5678
26
```

Method 1 - using str() and a for-loop

In [ ]:
```
# Read an integer as a string:
s = input("Enter an int number: ")

# for example, print 6 if a is 123, 18 if a is 369.
total = 0

None

print(total)
```

Method 2: Using indexing and a for-loop

In [ ]:
```
# Read an integer as a string:
s = input("Enter an int number: ")

# for example, print 6 if a is 123, 18 if a is 369.
total = 0
None

print(total)
```

Method 3: Using a while-loop

Do it for any number of digits and using while loop

In [1]:
```
# Read an integer:
n = int(input("Enter an int number: "))
# for example, print 6 if a is 123, 18 if a is 369.

total = 0
None

print(total)
```

```
0
```

Method 4: Using list comprehension

This topic will be handled later sessions.

Using built-in function `sum()` , `int()` , and list comprehension, we can just do it in one line of code.

- To use built-in `sum()` function, use `del` command to undefine sum which was previously defined.
- `Input()` returns a `str` type data.
- Use list comprehension and get each character.

- Use `sum()` to sum up the element in the list.

In [ ]:
```
# Read an integer as a string:
s = input("Enter an int number: ")

# for example, print 6 if a is 123, 18 if a is 369.

total = None
print(total)
```

for 반복문과 문자열 연산을 사용하기

for 반복문과 **문자열 연산** 을 사용하여 다음과 같이 출력하십시오. 문자열 연산을 사용하면, if 조건문 없이, 하나의 for 반복문으로 구현할 수 있습니다. `if` 조건문을 사용하지 않고 구현하십시오. 문자열 연산이란, 예를 들면, `a * 3` 은 `aaa` 를 나타냅니다.

Expected Output:

```
*
**
***
****
*****
******
*******
********
*********
**********
```

In [42]:
```
n = 11
None
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

for 반복문과 문자열 연산을 사용하기

for 반복문과 **문자열 연산** 을 사용하여 다음과 같이 출력하십시오. 문자열 연산을 사용하면, if 조건문 없이, 하나의 for 반복문으로 구현할 수 있습니다. 문자열 연산이란, 예를 들면, `a * 3` 은 `aaa` 를 나타냅니다.

Expected Output:

```
***********
**********
*********
```

```
********
*******
******
*****
****
***
**
*
```

In [37]:

```
n = 11
None
```

```
***********
**********
*********
********
*******
******
*****
****
***
**
*
```

for 반복문과 문자열 연산, if 조건문을 사용하기

for 반복문과 문자열 연산, if 조건문을 사용하여 다음과 같이 출력하십시오. 문자열 연산이란, 예를 들면, `a * 3` 은 `aaa` 를 나타냅니다. 하나의 for 반복문과 하나의 조건문이면 코딩이 가능합니다.

Expected Output:

```
*
**
***
****
*****
******
*******
********
*********
**********
***********
**********
*********
********
*******
******
*****
****
***
**
*
```

In [41]:

```
n = 22
None
```

```
*
**
***
****
*****
******
*******
********
*********
**********
***********
**********
*********
********
*******
******
*****
****
***
**
*
```

for 반복문과 문자열 연산, if 조건문을 사용하기

for 반복문과 문자열 연산을 사용하여 다음과 같이 출력하십시오. 필요하면, if 조건문을 사용해도 되지만, 문자열 연산을 사용하면, 조건문 없이 두 개의 for 반복문으로 가능합니다. 문자열 연산이란, 예를 들면, `a * 3` 은 `aaa` 를 나타냅니다.

Expected Output: h = 7

```
      *
     ***
    *****
   *******
  *********
 ***********
*************
 ***********
  *********
   *******
    *****
     ***
      *
```

```
h = 5
```

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

In [29]:
```
h = 7
```

```
None
```

```
      *
     ***
    *****
   *******
  *********
 ***********
*************
 ***********
  *********
   *******
    *****
     ***
      *
```

In [43]:
```
h = 5
None
```

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

구구단 출력하기

for와 range 함수를 사용하여 아래와 같이 구구단을 출력하십시오.

Expected Output:

```
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

Solution:

In [1]:
```
None
```

```
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

코인 가격의 변화

어떤 코인의 가격은 그야말로 동전 던지기와 비슷한 것 같습니다. 아주 위험한 일이죠. 매일 한 번 동전을 던져서 앞면이 나오면 전날 가격의 1.5배가 되고, 뒷면이 나오면 전날 가격의 절반이 됩니다. 1일에 주식의 가격이 1,024원이었을 때, 4일 주식의 가격이 나올 수 있는 경우를 모두 구하십시오. (힌트: for 반복문이 3개 중첩되어야 합니다)

Solution:

In [9]:
```
None
```

Out[9]:
```
{128.0, 384.0, 1152.0, 3456.0}
```

수열의 값 구하기

다음과 같은 수열이 있을 때 n번째 수열의 값을 구하십시오.

$$\begin{gathered} 1 \\ 1+(1+2) \\ 1+(1+2)+(1+2+3) \\ 1+(1+2)+(1+2+3)+(1+2+3+4) \\ \vdots \\ 1+(1+2)+(1+2+3)+(1+2+3+4)+\cdots+(1+2+\cdots+n) \end{gathered}$$

Sample Run:

```
if __name__ == '__main__':
    total = 0
    last_num = int(input('last_num: '))
    for i in range(1, last_num + 1):
        total += recursive_sum(i)
    print(total)
```

Expected Output:

```
last_num:  5
35
```

In [15]:
```
None
```

```
35
```

학습 정리

1. break 명령어 사용하기
2. Nested loops 이해하기
3. for문 & range()
4. Enumerate()

참고자료

- Python Iteration-for loop
- Python Iteration-while loop

"Everything is permissible" - but not everything is beneficial. "Everything is permissible" - but not everything is constructive. 1 Corinthians 10:23