Chapter 11 – Loops

Contents

Chapter-11 For Loops

ကုဒ်အစုအဝေး(block of code)တစ်ခုကို အကြိမ်ကြိမ် အထပ်ထပ်(repetitive) execute လုပ်လို သည့်အခါ for loop နှင့် while loop ကို အသုံးပြုသည်။ For loop ကို စုစုပေါင်းတန်ဖိုး(sum)၊ ပျမ်းမျှတန်ဖိုး (average) စသည့် aggregate များကို တွက်ရန် သုံးသည်။

ကုဒ်အစုအဝေး(block of code)တစ်ခုကို အကြိမ်ကြိမ် အထပ်ထပ်(repetitive) execution လုပ်နေ ခြင်းကို **iteration** ဟုခေါ်သည်။ လုပ်ရမည့် အကြိမ်အရေအတွက် အတိအကျ ရှိလျှင် definite iteration ဖြစ်သည်။ လုပ်ရမည့် အကြိမ်အရေအတွက် အတိအကျ မရှိလျှင် indefinite iteration ဖြစ်သည်။

Definite iteration သည် for loop ဖြစ်သည်။ Indefinite iteration သည် while loop ဖြစ်သည်။ ထို့ကြောင့် python တွင် for loop နှင့် while loop ဟူ၍ (၂)မျိုးရှိသည်။ For loop သည် definite iteration loop ဖြစ်သည်။ Loop ပတ်ရမည် အကြိမ်အရေအတွက် အတိအကျရှိသည့် loop ဖြစ်သည်။

While loop တွင် loop ပတ်ရမည် အကြိမ်အရေအတွက် အတိအကျ မရှိသောကြောင့် indefinite iteration ဟုခေါ်ခြင်းဖြစ်သည်။ While loop သည် အလိုရှိသည့် အခြေအနေတစ်ခုသို့ ရောက်သည့်တိုင်အောင် loop ကို ပတ်နေရမည်ဖြစ်သောကြောင့် အရေအတွက် အတိအကျ ပြောရန် မဖြစ်နိုင်ပါ။ (indefinite iteration ဖြစ်သည်။)

11.1.1 Collection-Based or Iterator-Based Loop (For Loop)

Loop တစ်ကြိမ်ပတ်တိုင်း variable(i)ဖြင့် အစုအဝေးတစ်ခု(a collection of objects)ထဲမှ itemများ တစ်ခုချင်းစီကို ထုတ်ယူပြီး loop အတွင်းသို့ထည့်က ဆောင်ရွက်ရမည့် ကိစ္စများ ဆောင်ရွက်စေခြင်းကို **Collection-Based Loop** သို့မဟုတ် **Iterator-Based Loop** ဟု ခေါ်သည်။

Python for loop ၏ syntax မှာ အောက်ပါအတိုင်း ဖြစ်သည်။ **for** keyword ၏ နောက်တွင် variable (i) လိုက်သည်။ ၎င်းနောက်တွင် **in** keyword လိုက်သည်။ နောက်ဆုံးတွင် <iterable> item များရှိသည်။

```
for <var> in <iterable>:
    <statement(s)>
```

<iterable>သည် object များ အစုအဝေး ဖြစ်သည်။(<iterable> is a collection of objects)။ ဥပမာ <iterable>သည် list တစ်ခု ဖြစ်နိုင်သလို tuple တစ်ခုလည်း ဖြစ်နိုင်သည်။ Loop body အတွင်းရှိ လုပ်ရမည့် **<statement(s)>**ကို indentation ဖြင့် ရေးရမည်။ Python control structure အရ for loop ထဲ၌ရှိသည့် item တစ်ခုချင်းစီကို execute လုပ်သည်။ ဥပမာ- i = 1 အတွက် execute လုပ်သည်။ ထို့နောက် i = 2 အတွက် execute လုပ်သည်။ စသည်ဖြင့် သတ်မှတ်ထားသည့် အရေအတွက် ရောက်သည့်တိုင်အောင် execute လုပ်သည်။

```
for i in <collection>
    <loop body>
```

11.1.2 Iterables

Python တွင် iterable ဆိုသည်မှာ loop ပတ်နိုင်သည့် data type များ ဖြစ်ကြသည်။ Iterable ၏ အဓိပ္ပာယ်မှာ iteration လုပ်နိုင်သည့် object များ ဖြစ်သည်။ (means an object can be used in iteration)။ Object တစ်ခုသည် iterable ဖြစ်လျှင် built-in Python function **iter()** ထဲသို့ ထည့်ပေးနိုင်သည်။ String , list, tuple, set, dict စသည့် data type များသည် iterable ဖြစ်သည်။

```
iter("python")                                          # String
```

```
<str_iterator at 0x208d9926088>
```

```
iter(('red', 'green', 'blue', 'yellow'))                #Tuple
```

```
<tuple_iterator at 0x208d99009c8>
```

```
iter(['red', 'green', 'blue', 'yellow'])                 # List
```

```
<list_iterator at 0x208d98e8148>
```

```
iter({'matthew': 'blue', 'rachel': 'green', 'raymond': 'red'})#dict
```

```
<dict_keyiterator at 0x208d9919228>
```

```
iter({'red', 'green', 'blue', 'yellow'})                #set
```

```
<set_iterator at 0x208d98ea0e8>
```

အောက်ပါတို့သည် iterable မဟုတ်သည့် သို့မဟုတ် iterable မဖြစ်သည့် object များ ဖြစ်ကြသည်။ အောက်က ဥပမာတွင် integer သည် iterable မဟုတ်သောကြောင့် TypeError ပေးလိမ့်မည်။

```
iter(42)
```

```
TypeError: 'int' object is not iterable
```

အောက်က ဥပမာတွင် float သည် iterable မဟုတ်သောကြောင့် TypeError ပေးလိမ့်မည်။

```
iter(3.1)
```

```
TypeError: 'float' object is not iterable
```

အောက်က ဥပမာတွင် len() သည် iterable မဟုတ်သောကြောင့် TypeError ပေးလိမ့်မည်။

```
iter(len)
```

```
TypeError: 'builtin_function_or_method' object is not iterable
```

အတိုချုပ်အားဖြင့် collection သို့မဟုတ် container အမျိုးအစား(type) များဖြစ်ကြသည့် string, list, tuple, dict, set, နှင့် frozenset တို့သည် iterable များ ဖြစ်ကြသည်။ Loop ပတ်နိုင်သည့် data type များ ဖြစ်ကြသည်။ integer နှင့် float တို့သည့် iterable data type များ မဟုတ်ပါ။

11.1.3 range() function

range() built-in function ကို for loop နှင့် တွဲ၍ အလွန်အသုံးများသည်။ Iterate လုပ်နိုင်သည့် ကိန်းအစဉ်အတန်း(sequence of integers)ကို အလိုရှိသည့်အခါ built-in **range()** function ကို သုံးနိုင်သည်။ **range()** ထုတ်ပေးသည့်အရာမှ object of class range တစ်ခုသာဖြစ်သည်။ List တစ်ခု သို့မဟုတ် tuple မဟုတ်ပါ။ Range object ကို iterate လုပ်နိုင်သည်။

```
for number in range(3):
    print(number, end = " ")    # end = " " သည် output ကို အလျားလိုက်ပေါ်စေရန်
```

```
0 1 2
```

range([start], stop, [step])တွင် start သည် စဆုံး item ဖြစ်သည်။ Stop သည် အဆုံး item ကို ရည်ညွှန်းသည်။ သို့သော် ထိုအဆုံးကိန်း မပါဝင်ပါ။ Step သည် အစနှင့် အဆုံး အကြား item မည်မျှကို ကျော်ရမည်ကို ဖော်ပြသည်။

- **start**: အစဆုံးကိန်း(starting number of the sequence)
- **stop**: အဆုံးကိန်း မတိုင်ခင်ကိန်း(generate numbers up to but not including this number)
- **step**: This is the difference between each number in the sequence.

Square brackets[ ]ထဲထည့်ရေးထားသည့် start နှင့် step သည် လိုအပ်သည့်အခါမှသာ ထည့်ပေးရန် လိုသည်။ မထည့်ပေးလျှင် star = 0 ဖြစ်သည်။ step = 1 ဖြစ်သည်။

```
for number in range(0, 8, 2):              # last one is step
    print(number, end = " ")
```

```
0 2 4 6
```

```
total = 0
for number in range(8):
    total += number
print(total)
```

```
28
```

11.2 Collection တစ်ခုကို Loop ပတ်ခြင်း(Looping over a Collection)

len() function ဖြင့် item အများ၏ အရေအတွက်ကို ရှာပြီး **range()** function ထဲထည့်၍ loop ပတ်နိုင်သည်။

```
colors = ['red', 'green', 'blue']
for i in range(len(colors)):
    print (colors[i])
```

```
red
green
blue
```

Len() နှင့် range()ကို မသုံးဘဲ `for color in colors` code တွင် colors ဆိုသည့် list ထဲမှ color item တစ်ခုချင်းစီကို ခေါ်ပြီး loop ပတ်နိုင်သည်။

```
for color in colors:
    print(color)
```

```
red
green
blue
```

```
for x in [(1, 2),(3, 4),(5, 6)]:
    print(x)
```

```
(1, 2)
(3, 4)
(5, 6)
```

Tuple တွေပါဝင်နေတဲ့ list တစ်ခုထဲက item တွေ တစ်ခုချင်းစီကိုလည်း for loop ပတ်ပြီးထုတ်ယူနိုင်သည်။

```
for i, j in [(1, 2), (3, 4), (5, 6)]:
    print(i, j)
```

```
1 2
3 4
5 6
```

Loop ကို ပြောင်းပြန်ပတ်ခြင်း(Looping backwards)

Loop ကို နည်း(၂)မျိုးဖြင့် ပြောင်းပြန်ပတ်နိုင်သည်။

(၁) **len()** နှင့် **range()**ကို သုံး၍ `start: -1, end:-1, step:-1` ပေး၍ loop ကို ပြောင်းပြန်ပတ်နိုင်သည်။

(၂) **reversed()** ကို သုံး၍ Loop ကို ပြောင်းပြန် ပတ်နိုင်သည်။

```
colors = ['red', 'green', 'blue']
for i in range(len(colors)-1, -1, -1):
    print(colors[i], end=' ')  # အလျားလိုက်ပေါ်စေရန် end=' ' ကို သုံးသည်။
```

```
blue green red
```

reversed() ကို သုံး၍ loop ကို ပြောင်းပြန် ပတ်နိုင်သည်။

```
for color in reversed(colors):
    print(color , end=' ')     # အလျားလိုက်ပေါ်စေရန် end=' ' ကို သုံးသည်။
```

```
blue green red
```

```
colors = ['red', 'green', 'blue']
index = len(colors) - 1
while index >= 0:
    print(colors[index], end=' ')
    index -= 1
```

```
blue green red
```

Item များနှင့် index နံပါတ်များကို တွဲ၍ ဖော်ပြရန် loop ပတ်ခြင်း(Looping over a collection and indices)

```
colors = ['red', 'green', 'blue', 'yellow']
for i in range(len(colors)):
    print (i, '--->', colors[i])
```

```
0 ---> red
1 ---> green
2 ---> blue
3 ---> yellow
```

အောက်တွင် **enumerate()** function ကို အသုံးပြုထားသည်။ **enumerate()** သည် index နှင့် item ကို တွဲပြီး tuple တစ်ခု ပြန်ထုတ်ပေးသည်။ ထို tuple ကို for loop ပတ်ပြီး တစ်တွဲချင်းစီ ထုတ်ယူသည်။ **enumerate()** အသုံးများသည့် function တစ်မျိုးဖြစ်သည်။

```
for i, color in enumerate(colors):
    print(i, '--->', color)
```

```
0 ---> red
1 ---> green
2 ---> blue
3 ---> yellow
```

11.3 Collection နှစ်ခုကို Loop ပတ်ခြင်း (Looping Over Two Collections)

Collection နှစ်ခု သို့မဟုတ် list နှစ်ခုကို အတွဲဖြစ်အောင် လုပ်၍ loop ပတ်နိုင်သည်။ Loop ပတ်သည့် အခါ အရေအတွက် အနည်းဆုံးရှိသည့် collection သို့မဟုတ် list အထိသာ ရသည်။ ထို့ကြောင့် **min()** ကို အသုံးပြု ထားသည်။ `min(len(names), len(colors))` ဖြင့် loop ပတ်ရမည့် အကြိမ် အနည်းဆုံး အရေအတွက်ကို ရှာသည်။

```
names = ['raymond', 'rachel', 'matthew']
colors = ['red', 'green', 'blue', 'yellow']
n = min(len(names), len(colors))
for i in range(n):
```

```
    print(names[i], '--->', colors[i])
```

```
raymond ---> red
rachel ---> green
matthew ---> blue
```

Parallel iteration လုပ်ရန်အတွက် **zip( )** function ကို အသုံးပြုထားသည်။

```
for name, color in zip(names, colors):
    print(name, '--->', color)
```

```
raymond ---> red
rachel ---> green
matthew ---> blue
```

11.4 Looping in Sorted Order

ယေရှာမှ ကြီးရာ သို့မဟုတ် စာလုံးရေ အနည်းရာမှ များရာသို့ စသည်ဖြင့် စီထားပြီး(sorted)သား ဖြစ်အောင် loop ပတ်ခြင်း

```
colors = ['red', 'green', 'blue', 'yellow']
# Forward sorted order
for color in sorted(colors):
    print(color)
```

```
blue
green
red
yellow
```

```
# Backwards sorted order
for color in sorted(colors, reverse=True):
    print(color)
```

```
yellow
red
green
blue
```

11.5 မိမိ အလိုရှိသည့် အတိုင်းဖြစ်အောင် စီခြင်း(Custom Sort Order)

```
colors = ['red', 'green', 'blue', 'yellow']
def compare_length(c1, c2):
    if len(c1) < len(c2): return -1
    if len(c1) > len(c2): return 1
    return 0
```

```
print(sorted(colors, key = len))
```

```
['red', 'blue', 'green', 'yellow']
```

11.6 Dictionary key များဖြင့် Loop ပတ်ခြင်း(Looping Over Dictionary Keys)

```
my_dict = {'matthew': 'blue', 'rachel': 'green', 'raymond': 'red'}
for k in my_dict:
    print(k)
```

```
matthew
rachel
raymond
```

```
my_dict = {'matthew': 'blue', 'rachel': 'green', 'raymond': 'red'}
for k in my_dict.keys():      # .key() ဖြင့် loop ပတ်သည်။
    print(k)
```

```
matthew
rachel
raymond
```

11.7 Dictionary Key နှင့် Value များဖြင့် Loop ပတ်ခြင်း(Looping over dictionary keys and values)

```
# Not very fast, has to re-hash every key and do a lookup
for k in my_dict:
    print(k, '--->', my_dict[k])
```

```
matthew ---> blue
rachel ---> green
raymond ---> red
```

```
# # .items()သည် key နှင့် value နှစ်ခုစလုံးကို ဆိုလိုသည်။
for k, v in my_dict.items():
    print(k, '--->', v)
```

```
matthew ---> blue
rachel ---> green
raymond ---> red
```

11.8 Looping Over Multiple Iterables

တစ်ခုထက်ပိုများသည့် iterable collection များဖြစ်သော list, tuple တို့ကို looping လုပ်သည့်အခါ **zip()** function ကို အသုံးပြုသည်။ **zip()** ကို အသုံးပြု၍ တစ်ခုထက် ပိုများသည့် list, tuple တို့ကို မည်ကဲ့သို့ looping လုပ်သည်ကို အောက်တွင် ရှင်းပြထားသည်။ **zip()** function မှ tuple များကိုသာ ထုတ်ပေးသည်။ Tuple ကို for loop ပတ်ရန် unpack လုပ်ရသည်။

```
letters = ['a', 'b', 'c']
numbers = [0, 1, 2]
special_char = ['@', '$', '!']
for l, n, sc in zip(letters, numbers, special_char):
    print(f'Letter: {l} Number: {n} Special Character: {sc}')
```

```
Letter: a Number: 0 Special Character: @
Letter: b Number: 1 Special Character: $
Letter: c Number: 2 Special Character: !
```

`l, n, sc in zip(letters, numbers, special_char)` တွင် letters ဟုသည့် list ထဲမှ l ဖြင့် item များကို ထုတ်ယူသည်။ numbers ဟုသည့် list ထဲမှ n ဖြင့် item များကို ထုတ်ယူသည်။ special_char ဟုသည့် list ထဲမှ sc ဖြင့် item များကို ထုတ်ယူသည်။ **zip()** ဖြင့် တွဲသည်။ (combine လုပ်သည်။) အောက်တွင် list (၄)ခုကို loop တစ်ခုတည်းဖြင့် တွဲနိုင်ပုံကို ဖော်ပြထားသည်။

```
letters = ['a', 'b', 'c']
numbers = [0, 1, 2]
special_char = ['@', '$', '!']
operators = ['*', '/', '+']
for l, n, sc, o in zip(letters, numbers, special_char, operators):
    print(f'Letter:{l} Number:{n}
            Special Character:{sc} Operator:{o}')
```

```
Letter: a Number: 0 Special Character: @ Operator: *
Letter: b Number: 1 Special Character: $ Operator: /
Letter: c Number: 2 Special Character: ! Operator: +
```

11.9 For Loop ကိုသုံး၍ Dictionary အတွင်းရှိ Item များကို ရေတွက်ခြင်း

အောက်က ဥပမာသည် list တစ်ခုထဲတွင် ပါဝင်နေသည့် item များ၏ ပါဝင်မှု အကြိမ် အရေအတွက်ကို ရှာသည့် ဥပမာ ဖြစ်သည်။ d ဆိုသည့် empty dict တစ်ခုဆောက်သည်။ if statement ဖြင့် d ထဲတွင် မရှိလျှင်(True ဖြစ်လျှင်) ထည့်မည်။ (ရှိနေလျှင်(False) ဘာမှ မလုပ်ပါ။) တစ်ခါတွေ့တိုင်း (၁) တိုးလိမ့်မည်။

```
colors = ['red', 'green', 'red', 'blue', 'green', 'red']
# Simple, basic way to count. A good start for beginners.
d = {}
for color in colors:
    if color not in d:     # ထဲတွင် မရှိလျှင်(True)  ထည့်မည်။ ရှိနေလျှင် ဘာမှ မလုပ်ပါ။
        d[color] = 0
    d[color] += 1          # တစ်ခါတွေ့တိုင်း (၁) တိုးလိမ့်မည်။
print(d)
```

```
{'red': 3, 'green': 2, 'blue': 1}
```

အထက်မှ code များကို ဖြင့် pythonic စတိုင် ပြန်ရေးထားသည်။

```
d = {}
for color in colors:
    d[color] = d.get(color, 0) + 1
print(d)
```

```
{'red': 3, 'green': 2, 'blue': 1}
```

11.10 နာမည်၏ အက္ခရာအလုံးရေကို လိုက်၍ Group ဖွဲ့ခြင်း(Grouping with dictionaries)

ပထမဦးစွာ empty dictionary တစ်ခု ပြုလုပ်သည်။ names ဆိုသည့် dictionary ထဲမှ နာမည် တစ်လုံချင်းစီကို loop ပတ်သည်။ နာမည်တစ်လုံချင်းစီ၏ အရှည်ကို key ဖြစ်ထားပြီး ထို key အတွက် အရှည်တူသည့် နာမည်များကို list တစ်ခု ပြုလုပ်သည်။

```
names = ['raymond', 'rachel', 'matthew', 'roger',
         'betty', 'melissa', 'judith', 'charlie']
# In this example, we're grouping by name length
d = {}
for name in names:
    key = len(name)      # နာမည်စာလုံးမှ အက္ခရာအလုံးရေကို တွက်သည်။
    if key not in d:     # အက္ခရာအလုံးအရေအတွက်သည် d ထဲတွင် မရှိလျှင် ထည့်ရန်
        d[key] = []
    d[key].append(name)
d
```

```
{7: ['raymond', 'matthew', 'melissa', 'charlie'],
 6: ['rachel', 'judith'],
 5: ['roger', 'betty']}
```

String များကို ပေါင်းစပ်(concatenate)ရန် အတွက်လည်း သုံးသည်။

```
names=['raymond', 'rachel', 'matthew', 'roger', 'betty', 'melissa',
       'judith', 'charlie']
s = names[0]
for name in names[1:]:
    s += ', ' + name
print(s)
print(names)
```

```
raymond, rachel, matthew, roger, betty, melissa, judith, charlie
```

```
['raymond', 'rachel', 'matthew', 'roger', 'betty', 'melissa', 'judit
h', 'charlie']
```

Sequence များကို update လုပ်ရန်အတွက်လည်း သုံးသည်။ 0 မှ 9 အထိ ကိန်းများကို တစ်လုံးချင်းစီ နှစ်ထပ်ကိန်းတင်၍ အားလုံးကို ပေါင်းသည်။

```
result = []
for i in range(10):
    s = i ** 2                          # တစ်လုံးချင်းစီ နှစ်ထပ်ကိန်းတင်သည်။
    result.append(s)
print(sum(result))
```

```
285
```

11.11 List ကို Loop ပတ်ခြင်း(Looping lists)

```
my_list = [1, 2, 'Python', {'a':1, 'b':2}]
for item in my_list:
    print(item)1
```

```
1
2
Python
{'a': 1, 'b': 2}
```

enumerate()

Index နံပါတ်များဖြင့်လည်း ပတ်နိုင်သည်။

```
for idx, val in enumerate(my_list):
    print('idx: {}, value: {}'.format(idx, val))
```

```
idx: 0, value: 1
idx: 1, value: 2
idx: 2, value: Python
idx: 3, value: {'a': 1, 'b': 2}
```

11.12 Looping Dictionaries

```
my_dict = {'hacker': True, 'age': 72, 'name': 'John Doe'}
for val in my_dict:
    print(val)
```

```
hacker
age
name
```

```
for key, val in my_dict.items():
    print('{}={}'.format(key, val))
```

```
hacker=True
age=72
name=John Doe
```

သုညမှ (၉)အထိ ကိန်း(၁၀)လုံးမှ အစဉ်လိုက် ကိန်း(၅)လုံးစီပါသည့် list နှစ်ခု တည်ဆောက်ပါ။

Empty list နှစ်ခုတည်ဆောက်သည်။ `range(10)` ကို for loop ပတ်သည်။ အလယ်ကိန်း (midpoint) (၅)ထက် ငယ်လျှင် lower ဆိုသည့် list ထည့်သည်။ ကြီးလျှင် upper ဆိုသည့် list ထဲထည့်သည်။

```
midpoint = 5                          # set the midpoint
lower = []; upper = []                # make two empty lists


for i in range(10):
    if (i < midpoint):      # split the numbers into lower and upper
        lower.append(i)
    else:
        upper.append(i)
print("lower:", lower)
print("upper:", upper)
```

```
lower: [0, 1, 2, 3, 4]
upper: [5, 6, 7, 8, 9]
```

11.13 Breaking Out of Loops (break , continue and pass statement)

Loop များအတွက် statement (၃)ခုမှာ

- **break** statement
- **continue** statement နှင့်
- **pass** statement တို့ဖြစ်သည်။

Break statement နှင့် continue statement တို့ကို သုံး၍ loop ပတ်ပုံ(loop behavior)ကို ပြောင်းနိုင်သည်။ Break statement ကို သုံး၍ while loop တွင် ရပ်တန့်သကဲ့သို့ loop တွင်းလည်း ရပ်တန့်ပစ်နိုင်သည်။ တစ်နည်းအားဖြင့် loop အတွင်းမှ exit လုပ်သည်။ Continue statement ကို သုံး၍ while loop တွင် ကျော်သွားသကဲ့သို့ loop တွင်းလည်း next iteration သို့ကျော်သွားနိုင်သည်။

break: Jumps out of the closest enclosing loop (past the entire loop statement)

```
#Stop the execution of the loop.
my_list = [1, 2, 3, 4, 'Python', 'is', 'neat']
for item in my_list:
    if item == 4:
        break
```

```
    print(item)
```

```
1
2
3
```

```
for i in ['green', 'yellow', 'blue', 'pink']:
    if 'blue' in i: # 'blue' ရောက်သည့်အချိန်တွင် loop ကို လုံးဝ ရပ်တန့် ပစ်သည်။
        break       # 'blue' မှ စပြီး loop ထဲမှ လုံးဝ ထွက်သွားသည်။
    print(i)
```

```
green
yellow
```

Continue statement ကို လက်ရှိပတ်နေသည့် loop ကို ရပ်ပစ်ပြီး နောက်လာမည့် item အတွက် ဆက်(continue)ပတ်လိုသည့်အခါတွင် သုံးသည်။ (continue statement terminates the current iteration and proceeds to the next iteration)။ အောက်ဥပမာတွင် 'blue' ရောက်သည့်အခါ loop ကို ရပ်ပစ်ပြီး ပြန်စ,သောကြောင့် 'blue' မပါတော့ပေ။

Continue statement

continue: ရောက်သည့်အခါ လက်ရှိရောက်နေသည့် နေရာမှ ရပ်တန့်ပြီး နောက်တစ်ပတ်ကို ဆက်ပတ်သည်။ (Jumps to the top of the closest enclosing loop (to the loop's header line))

```
for i in ['green', 'yellow', 'blue', 'pink', 'red']:
    if 'blue' in i:      # 'blue' ရောက်သည့်အချိန်တွင် loop ရပ်သွားလိမ့်မည်။
        continue
    print(i)
```

```
green
yellow
pink
red
```

အထက်ဥပမာတွင် 'blue' ရောက်သည့်အချိန်တွင် `i='blue'` အပတ်ကို ရပ်တန့်ပြီးနောက်ထပ် အပတ်အသစ်(next iterable item)ကို ပြန်စပတ်သည်။ ထို့ကြောင့် 'blue' item မပါတော့ပေ။

```
# Continue to the next item without executing the lines
my_list = [1, 2, 3, 4, 'Python', 'is', 'neat']
for item in my_list:
    if item == 1:       # item သည် 1 ဖြစ်လျှင် loop ရပ်ပြီးနောက် တစ်ခု စမည်။
        continue
    print(item)
```

```
2
3
4
Python
is
neat
```

11.13.1 “pass” Statement

ပြင်ပအခြေအနေတစ်ခုခုမှ loop ရပ်တန့်ရမည့်အခါ **pass** statement ကို ထည့်ထားခြင်းဖြင့် loop ပုံမှန် ဆက်ပတ်နိုင်သည်။ ကုဒ်များ ပြီးဆုံးသည်အထိ ဆက် run နိုင်သည်။ **pass** statement ကို conditional if statement နှင့် တွဲသုံးလေ့ရှိသည်။

pass: Does nothing at all: it’s an empty statement placeholder

```
for number in range(1,5):
    if number == 2:        # If the number is 2, proceed as normal
        pass
    print(number)
```

```
1
2
3
4
```

11.13.2 For loop နှင့် else Clause တွဲသုံးနိုင်သည်

Iterable item များ မရှိတော့၍ loop အဆုံးသတ်(terminate)သည့်အခါ else clause သည် စတင် အလုပ် လုပ်လိမ့်မည်။ Loop တစ်ခုသည် ပြီးဆုံးသည် သို့မဟုတ် မပြီးခင် ရပ်တန့်သွားသည် သိရန်တွက် for loop တွင် else clause ကို ထည့်သုံးသည်။

Loop else block: Runs if and only if the loop is exited normally (i.e., without hitting a break)

```
for i in ['green', 'yellow', 'blue', 'pink']:
    print(i)
else:
    print('Done.')  # Will execute
```

```
green
yellow
blue
pink
Done.
```

Break statement ကြောင့် သည် အဆုံးထိ ရောက်အောင် ပတ်ခွင့် မရဘဲ ရပ်တန့်သွားလျှင် else clause သည် အလုပ်လုပ်လိမ့်မည် မဟုတ်ပေ။ (The else clause won’t be executed if the list is broken out of with a break statement) အောက်တွင် ဥပမာဖြင့် ပြထားသည်။

```
for i in ['green', 'yellow', 'blue', 'pink']:
```

```
    if i == 'blue':
        break
    print(i)
else:
    print('Done.')   # blue ရောက်သည့်အခါ break ဖြစ်သွားပြီး loop ရပ် သွားသည်။
```

```
green
yellow
```

`'blue'` ရောက်သည့်အခါ break ဖြစ်သွားပြီး loop ရပ်တန့်ပြီး လုံးဝ loop အပြင်ဘက်သို့ ထွက်သွားသည်။ Loop ပြီးအောင် မပတ်နိုင် သောကြောင့် `else:` ဆီသို့ မရောက်ပါ။

11.14 Nested Loops

အောက်က ကုဒ်ကို run ရင် ငယ်ငယ်တုန်းက ကျက်ခဲ့ရတဲ့ အလီစာရွက်ကလေးကို သတိရလိမ့်မည်။

```
for i in range(1,13):
    for j in range(1,13):
        print(f'{i} x {j} = {i*j}')
    print("--------------")
```

Output သိချင်ရင် run ကြည့်ပါ။

11.15 While Loop

While loop သည် စတင် အလုပ်လုပ်ရန် ပထမဦးစွာ အခြေအနေ(initial condition)တစ်ခုကို စစ်သည်။ True ဖြစ်လျှင် loop ကို စတင်အလုပ်လုပ်(loop starts executing)သည်။ Loop တစ်ပတ်ပြီးတိုင်း နောက်တစ်ခေါက် ပြန်စရန် while statement အခြေအနေ(test condition)ကို စစ်ဆေးပြန်သည်။ သတ်မှတ်ထားသည့် အခြေအနေ တစ်ခုသည် မှန်နေသမျှ ကာလပတ်လုံး while loop သည် မရပ်တန့်ဘဲ အလုပ်လုပ်နေလိမ့်မည်။ (As long as the condition remains true, the loop keeps executing)။ အခြေအနေ(test condition) False ဖြစ်သည့်အခါတွင် while loop ရပ်တန့်လိမ့်မည်။

သတိပြုရန်အချက်မှာ မိမိထည့်ပေးသည့် အခြေအနေ(test condition)သည် မည်သည့်အခါမျှ False မဖြစ်လျှင် while loop သည် ဘယ်တော့မှ ရပ်တန့်လိမ့်မည် မဟုတ်ပေ။ ထိုသို့ while loop exit မဖြစ်ခဲ့လျှင် Ctrl+C သို့မဟုတ် တစ်နည်းနည်းဖြင့် ပရိုဂရမ်ကို ရပ်တန့်ပါ။(On most systems, Ctrl-C will abort a running program.)

- While statement တွင် စစ်ရန် အခြေအနေ(test condition)တစ်ခု ပါဝင်ရမည်။
- While loop စတင်အလုပ် လုပ်ရန် True ဖြစ်သည့် အခြေအနေတစ်ခု လိုအပ်သည်။(Every while loop needs an initial condition that starts out true.)

- True ဖြစ်လျှင် while loop အတွင်းရှိ ကုဒ်များအားလုံး run လိမ့်မည်။ loop ၏ အောက်ဆုံးအထိ ရောက်အောင် run ပြီး နောက်တစ်ပတ် ပတ်ရန် while statement ဆီသို့ ပြန်တက်လာသည်။
- While statement ၏ test condition သည် False ဖြစ်လျှင် while loop ရပ်တန့်ပြီး loop မှ ထွက်(exit) သွားသည်။

  အောက်တွင် ဥပမာအဖြစ် while loop ရှိ ကုဒ်များကို တစ်ကြောင်းချင်းစီ ရှင်းပြထားသည်။

```
x = True                    # Assign True to x (x contains True)
while x == True:          # Evaluate the expression (evaluates to True)
    print("Hello")        # Print "Hello" to the terminal window
    x = False             # Assign False to x (x contains False)
                        # We've reached the end so re-evaluate condition
print("x is now false")
```

While loop ၏ အောက်ဆုံး အကြောင်းသို့ရောက်သည့်အခါ နောက်တစ်ကြိမ် ထပ်ပတ်ရန် while statement သို့ ပြန်တက်သွားသည်။ while statement ကို ပြန်စစ်သည်။ True မဖြစ်ပါက(fail ဖြစ်ပါက) loop မှ ထွက်သည်။

```
x = True
while x == True:  # Evaluate the expression (it fails, so exit loop)
    print("Hello")# (If it didn't we'd go around the loop again.)
    x = False
print("x is now false")
```

```
Hello                    → "Hello" print ထုတ်ပြီး နောက်တစ်ကြောင်း၌ False ဖြစ်သွားသည်။
x is now false
```

While loop တွင် ပတ်ရမည့် အကြိမ်အရေအတွက် အတိအကျ မရှိသောကြောင့် infinite loop အမျိုးအစား ဖြစ်သည်။ " i " သည် (၆)ထက် ငယ်နေသမျှ ကာလပတ်လုံး print ထုတ်ပေးမည့် while loop ဖြစ်သည်။

```
i= 0
while i< 6:
    i += 1
    print(i, end='')        # end =''သည် output များကို အလျားလိုက် ဖော်ပြပေးရန်
```

```
1 2 3 4 5 6
```

a သည် b ထက် ငယ်နေသမျှ ကာလပတ်လုံး while loop အလုပ် လုပ်နေလိမ့်မည်။ Loop တစ်ပတ် ပတ်ပြီးတိုင်း ကောင်တာအဖြစ် a ကို (၁)တိုးပေးသည်။ (၁၀)ပတ်ပြည့်ပြီးလျှင် a သည် b ထက် မငယ်တော့ပေ။ ထို့ကြောင့် test condition သည် False ဖြစ်သောကြောင့် loop ရပ်သွားလိမ့်မည်။

```
a, b = 0, 10 # test condition အဖြစ်သုံးရန် a နှင့် b ကို assign လုပ်သည်။
while a < b:
    print(a, end=' ')
    a += 1          # loop တစ်ပြတ်ပြီးတိုင်း ကောင်တာအဖြစ် a ကို (၁)တိုးပေးသည်။
```

```
0 1 2 3 4 5 6 7 8 9
```

ပေးထားသည့် list တစ်ခုမှ အနုတ်ကိန်းများကိုသာ ရွေးချယ်၍ list တစ်ခု တည်ဆောက်ပါ။ အပေါင်းကိန်းများနှင့် သုညကို ချန်ထားခဲ့ပါ။

```
list=[2, -16, 2, -5, 0, 1, -2, -3]
new_list=[]                 # empty list အသစ်တစ်ခု ပြုလုပ်သည်။
i=0
while i<len(list):          # list ထဲရှိ ကိန်းအရေအတွက်len(list)အတိုင်း loop ပတ်သည်။
    if (list[i]<0):         # အနုတ်ကိန်း ဖြစ်လျှင် new_list ထဲထည့်သည်။
        new_list.append(list[i])
    i+=1
print(new_list)
```

```
[-16, -5, -2, -3]
```

ထည့်ပေးသည့် string တစ်ခုမှ character တစ်လုံးချင်းစီ ပါဝင်သည့် list တစ်ခု ပြုလုပ်ပါ။

```
my_list=[]
my_str= str(input("Enter any word : "))
i=0
while i<len(my_str):          # my_str ၏ length အတိုင်း loop ပတ်မည်။
    my_list.append(my_str[i])
    i+=1
print(my_list)
```

```
Enter any word : Myanmar
['M', 'y', 'a', 'n', 'm', 'a', 'r']
```

ဥပမာ - ဂိမ်းတစ်ခုတွင် ကစားသူ(player)သည် ပါဝါ ရှိနေသမျှ ကာလပတ်လုံး ဆက်ကစားခွင့်ရှိသည်။ ပါဝါ မရှိတော့လျှင် Game Over ဖြစ်သည်။ တစ်ခါ ရှုံးတိုင်း ပါဝါတစ်ခု အနုတ်ခံရမည်။ စပေးသည့် ပါဝါမှာ (၃) ဖြစ်သည်။ (၃)ခါ ရှုံးခွင့်ရှိသည်။

power = 3 → စပေးသည့် ပါဝါမှာ (၃) ဖြစ်သည်။ ကစားသူသည် ပါဝါ ရှိနေသမျှ ကာလပတ်လုံး ဆက်ကစားခွင့် ရှိသည်။

```
power = 3
while power > 0:
    print("You are still playing, because your power is %d." %power)
    power = power - 1
print("\nOh no, your power dropped to 0! Game Over.")
```

```
You are still playing, because your power is 3.
You are still playing, because your power is 2.
You are still playing, because your power is 1.
Oh no, your power dropped to 0! Game Over.
```

နာမည်များကို တစ်ခုပြီးတခု လက်ခံပြီး list တစ်ခု တည်ဆောက်ပေးရန် နှင့် 'quit' စကားလုံးကို ရိုက်ထည့်လိုက်ပါက while loop မှ exit လုပ်ရန် ဥပမာ-

```
names = []           # Create a list of
new_name = ''        # Define a new_name variable other than 'quit'.
While new_name != 'quit': #'quit' မထည့်ပေးလျှင် ဆက်အလုပ်လုပ်နေလိမ့်မည်။
    # Ask the user to enter the name
    new_name=input("Tell me someone I should know OR enter'quit':")
    # Add this name to the list that was created earlier by
    names.append ( new_name )
print(names)          # ထည့်ပေးသမျှ အားလုံးကို print ထုတ်ပေးလိမ့်မည်။
```

```
Please tell me someone I should know OR enter 'quit':Yangon
Please tell me someone I should know OR enter 'quit':Mandalay
Please tell me someone I should know OR enter 'quit':quit
['Yangon', 'Mandalay', 'quit']
```

အပေါ်က ကုဒ်များ အားလုံး အလုပ် လုပ်သော်လည်း 'quit' ဆိုတဲ့ စာလုံးကိုပါ ထည့်၍ print ထုတ်ပေးသည်။ 'quit' သည် loop မှ ထွက်ရန် command ဖြစ်သောကြောင့် print ထုတ်ပေးသည့်ထဲတွင် ဖော်ပြရန် မလိုပါ။ အောက်တွင် အနည်းငယ်ပြင်ဆင် ရေးသည်။ အောက်တွင် အနည်းငယ်ပြင်ဆင် ရေးသည်။ names ဆိုသည့် list ဆောက်ပြီး နာမည်အားလုံးကို ထည့်သည်။ 'quit' ကိုပါထည့်ပြီး print မလုပ်ခင် နောက်ဆုံး item ဖြစ်သည့် 'quit' ကို delete လုပ်သည်။

```
names=[]
```

```
new_name = ''
while new_name != 'quit':   #'quit' မထည့်ပေးလျှင် ဆက်အလုပ်လုပ်နေလိမ့်မည်။
    new_name=str(input("Please tell me someone I should know,or enter
                'quit ':"))
    names.append(new_name)
del names[-1]                       #နောက်ဆုံး item ကို delete လုပ်သည်။
print(names)
```

```
Please tell me someone I should know, or enter 'quit':Yangon
Please tell me someone I should know, or enter 'quit':Mandalay
Please tell me someone I should know, or enter 'quit':quit
['Yangon', 'Mandalay']
```

While loop တွင် ပတ်ရမည့် အကြိမ် အရေအတွက် အတိအကျ မရှိသောကြောင့် infinite loop အမျိုးအစား ဖြစ်သည်။ တစ်ခါတစ်ရံ ဘယ်တော့မှ မရပ်တန့်သည် loop မျိုးရေးမိတတ်သည်။ (often endless cycles occur.)

အောက်က ကုဒ်လိုမျိုးရေးမိရင် ဒုက္ခရောက်မှာပါ။ Current_number ကို 1 လို့ assign လုပ်သည်။ While ကို current_number က 5 ညီရင် ရပ်မည်လို့ ဆိုပြီး test condition ပေးထားသည်။ Loop body ထဲမှာ တစ်ပတ်ပြီးတိုင်း current_number ကို (၁)တိုးဘို့ မေ့ခဲ့တယ်ဆိုရင် current_number က အမြဲတမ်း 1 ဖြစ်နေပြီး test condition က အမြဲ True ဖြစ်နေလို့ while loop က ဘယ်တော့မှ ရပ်မှာ မဟုတ်ပါဘူး။

```
current_number  =  1
# Count up to 5, printing the number each time.
while  current_number  <=  5 :
    print (current_number )
```

`current_number = current_number+1` သို့မဟုတ် `current_number += 1` ကောင်တာကို update လုပ်သည့် code တစ်ကြောင်းထည့်ပေးရမည်။

```
current_number  =  1
# Count up to 5, printing the number each time.
while current_number  <=  5 :
    print ( current_number, end = ' ')
    current_number  +=  1
```

```
1 2 3 4 5
```

11.16 Loop Controls (Using Break)

`break` သည် loop ကို control လုပ်သည့် keyword ဖြစ်သည်။ `break` ကို တွေ့သည်နှင့် တပြိုင်နက် loop ကို လုံးဝ ရပ်တန့်လိုက်သည်။ i သည် 3 နှင့် ညီပြီးနောက် loop ကို ရပ်တန့်ရန်

```
i= 0
```

```
while i< 6:
    print(i , end =" ")
    if i ==3:
        break                  # i ==3 ဖြစ်သည့်အခါ Loop ကို ရပ်တန့်ရန်
    i += 1
```

```
0 1 2 3
```

```
while 1:
    name = input ('Enter name:')
    if name == 'quit':
        break
    age = input('Enter age:')
    print ('Hello',name, '=>', int(age)**2)
```

```
Enter name:Jupyter
Enter age:25
Hello Jupyter => 625
Enter name:quit
```

While Loop နှင့် break တွဲသုံးသည့် ဥပမာ(၁)

```
word = 'python'
i = 0
while i < len(word):
    letter = word[i]
    i += 1
    if letter == 'o':              # 'o' ရောက်လျှင် Loop ကို ရပ်တန့်ရန်
        break                      # stop iter

    print(letter, end = " ")
```

```
p y t h
```

While Loop နှင့် break တွဲသုံးသည့် ဥပမာ(၂)

user name ထည့်ပေးသည့်အခါ character (၂)ခု အနည်းဆုံးပါပြီး printable character နှင့် alphabetic character များသာ လက်ခံမည်။ ထိုအချက်များနှင့် မကိုက်ညီပါက လက်မခံပါ။

`len(name) >= min_length` သည် name တွင် အနည်းဆုံး character (၂)ခု ပါရမည်။ ထို character (၂)ခု printable character များ ဖြစ်ရမည်။ (`name.isprintable()`)။ ထို character (၂)ခု alphabetic character များ ဖြစ်ရမည်။ (`name.isalpha()`)။ အချက်အားလုံးနှင့် ကိုက်ညီရမည် ဖြစ်သောကြောင့် **and** ဖြင့် ဆက်ထားသည်။

```
min_length = 2      # character (၂)ခု အနည်းဆုံး ပါရမည်။


while True:
    name = input('Please enter your name:')      # input လက်ခံသည်။
    if len(name) >= min_length and name.isprintable() and \
       name.isalpha():
        break                   # သင့်လျော်သည့်နာမည်ရလျှင် loop ကို ရပ်ပစ်မည်။
print('Hello, {0}'.format(name))
```

```
Please enter your name:a                ( character တစ်ခုတည်းကြောင့်လက်မခံပါ။)
Please enter your name:23               ( digit ဖြစ်သောကြောင့် လက်မခံပါ။)
Please enter your name:Python
Hello, Python
```

11.17 Using Continue

Continue သည် လုံးဝ loop ကို ရပ်တန့်လိုက်ခြင်းမဟုတ်ပါ။ လက်ရှိ ပတ်နေသည့် iteration ကို ကျော်(skip) သွားသည့် သဘောမျိုးဖြစ်သည်။ အောက်က ဥပမာတွင် `i==3` true ဖြစ်လျှင် လက်ရှိ ပတ်နေသည့် iteration ကျော်(skip လုပ်မည်။)သွားမည်။ True ဖြစ်သည့် iteration ကို ကျော်(skip)သွား loop ကို ပြီးဆုံးသည်အထိ ဆက်ပတ်မည်။

```
# if i ==3: Continue သည် (၃) ရောက်လျှင် loop ကို ရပ်တန့်ပြီး ပြန်ဆက်ပတ်ရန်
i= 0
while i < 7:
    i += 1
    if i==3:    # (၃)ရောက်လျှင် loopကို ရပ်တန့်ပြီး ပြန်ဆက်ပတ်သောကြောင့် (၃) မပါတော့ပါ။
        continue
    print(i, end= ' ')
```

1 2 4 5 6 7

`if letter == 't'` သည် 't' ရောက်လျှင် ရပ်တန့်ပြီး ပြန်ဆက်ပတ် သောကြောင့် 't' မပါတော့ပါ။

```
word = 'python'
i = 0
while i < len(word):
    letter = word[i]
    i += 1
    if letter == 't':
        continue       #'t'ရောက်လျှင် ရပ်တန့်ပြီး ပြန်ဆက်ပတ်သောကြောင့် 't'မပါတော့ပါ။
    print(letter, end = " ")
```

```
print("\nno 't'")
```

```
p y h o n
no 't'
```

`x%2 != 0` သည် သုညမကြွင်းလျှင်(မကိန်း ဖြစ်လျှင်)ရပ်တန့်ပြီး ပြန်ဆက်ပတ်သောကြောင့် မကိန်းများ မပါတော့ပါ။

```
x = 10
while x:
    x = x - 1
    if x % 2 != 0:     # သုည မကြွင်းလျှင် (မကိန်း ဖြစ်လျှင်) ရပ်တန့်ပြီး ပြန်ဆက်ပတ်ရန်
        continue       # Odd? - skip output
    print(x , end = " ")
```

```
8 6 4 2 0
```

True ဖြစ်သည့် iteration ကို ကျော်(skip)သွား loop ကို ပြီးဆုံးသည်အထိ ဆက်ပတ်မည်။

```
a = 0
while a < 10:  # 9 အထိ ပတ်မည်။
    a += 1
    if a % 2:  # စုံကိန်းကို ကျော်
        continue
    print(a, end =" ")
```

```
2 4 6 8 10
```

```
a = 0
while a < 10:  # 9 အထိ ပတ်မည်။
    a += 1
    if a%2 == 0: # မကိန်းကို ကျော်
        continue
    print(a, end =" ")
```

```
1 3 5 7 9
```

11.18 Using pass

```
word = 'python'
i = 0
while i < len(word):
    letter = word[i]
    i += 1
    if letter == 't':
        pass                          # do nothing
    print(letter, end = " ")
```

```
p y t h o n
```

11.19 ဖိုင်တစ်ခု အတွင်းရှိ စာလုံး(word) များကို ရေတွက်ခြင်း ဥပမာ

txt ဖိုင် အတွင်းမှ အမျိုးမတူသည့် စာလုံး(word)များကို ရေတွက်ပါ။ ထိုနောက် စာလုံး(၁)လုံး ထိုဖိုင်ထဲတွင် အကြိမ်ရေ မည်မျှပါဝင်သည်ကိုလည်း ရေတွက်ပြပါ။

Same directory ဖိုင်တစ်ခုကို read only ဖြင့် ဖွင့်သည်။

- ထိုဖိုင်အတွင်းရှိ လိုင်းများကို `.readlines()` ဖြင့် ဖတ်ယူသည်။ အစနှင့် အဆုံးရှိ space (leading and the trailing characters, space)ကို **strip()** method ဖြင့် ဖယ်ထုတ်သည်။ `.split(" ")` ဖြင့် space ရှိသည့် နေရာကို အလယ်နေရာအဖြစ်ထားပြီး စာလုံး(word)များကို ခွဲထုတ်သည်။

- စာလုံး(word)ကို key အဖြစ်ထားပြီး အရေအတွက် ကို value ဖြစ်ထည့်သိမ်းဆည်းမည့် empty dictionary တစ်ခုတည် ဆောက်သည်။ `word_count_dict = {}`
- စာလုံး(word) အားလုံးကို တစ်ခုချင်းစီ ပတ်သည်။ ရှိပြီးသား စာလုံး(word)ဖြစ်လျှင် (၁)တိုးသည်။ print လုပ်သည်။

```
input_file = open("mathematics_wiki.txt", 'r')
all_words = input_file.readlines()[0].strip().split(" ")
word_count_dict = {}                    # {'word1': n1, 'word2': n2}

for word in all_words:                   # word အားလုံး အတွက် loop ပတ်သည်။
    if word in word_count_dict:
        word_count_dict[word] += 1       # word_count_dict ထဲရှိလျှင် (၁)တိုးသည်။
    else:
        word_count_dict[word] = 1

print(word_count_dict)
print(len(word_count_dict))
```

```
{'Mathematics': 2, 'is': 3, 'the': 5, 'study': 1, 'of': 5, 'numbers,
': 1, 'quantity,': 1, 'space,': 1, 'structure,': 1, 'and': 7, 'chang
e.': 1, 'used': 1, 'throughout': 1, 'world': 1, 'as': 3, 'an': 1, 'e
ssential': 1, 'tool': 1, 'in': 3, 'many': 1, 'fields,': 2, 'includin
g': 1, 'natural': 1, 'science,': 1, 'engineering,': 1, 'medicine,':
1, 'social': 1, 'sciences.': 1, 'Applied': 1, 'mathematics,': 3, 'br
anch': 1, 'mathematics': 3, 'concerned': 1, 'with': 1, 'application'
: 2, 'mathematical': 3, 'knowledge': 1, 'to': 2, 'other': 1, 'inspir
es': 1, 'makes': 1, 'use': 1, 'new': 2, 'discoveries': 1, 'sometimes
': 1, 'leads': 1, 'development': 1, 'entirely': 1, 'disciplines,': 1
, 'such': 1, 'statistics': 1, 'game': 1, 'theory.': 1, 'Mathematicia
ns': 1, 'also': 1, 'engage': 1, 'pure': 3, 'or': 1, 'for': 2, 'its':
 1, 'own': 1, 'sake,': 1, 'without': 1, 'having': 1, 'any': 1, 'mind
.': 1, 'There': 1, 'no': 1, 'clear': 1, 'line': 1, 'separating': 1,
'applied': 1, 'practical': 1, 'applications': 1, 'what': 1, 'began':
 1, 'are': 1, 'often': 1, 'discovered.': 1}
79
```

'Mathematics' ဆိုသည့် စကာလုံး(word) နှစ်လုံးပါသည်။ 'is' ဆိုသည့် စကာလုံး(word) 3 လုံးပါသည်။ စသည်ဖြင့် ရေတွက်ပြီး dict တစ်ခု တည်ဆောက်ပေးသည်။ စုစုပေါင်း အမျိုးအစားမတူသည့် စာလုံး (၇၉)လုံး ပါဝင်သည်။

-End-

Reference: https://realpython.com/python-while-loop/