For loop

The `for` loop

- **`for` loop: Repeats a set of statements over a group of values.**
  - Syntax:

```
for target in object:
    statements
```

    - We indent the statements to be repeated with tabs or spaces.
    - **`target`** gives a name to each value, so you can refer to it in the ***statements***.
    - **`object`** can be a range of integers, string, list, ….

  - Example:

```
for x in [1, 2, 3, 4, 5]:
    print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

x	x * x
1	1
2	4
3	9
4	16
5	25

for loop

```
for target in object:          # Assign object items to target
    statements                 # Repeated loop body: use target
else:                          # Optional else part
    statements                 # If we didn't hit a 'break'
```

```
for target in object:          # Assign object items to target
    statements
    if test: break             # Exit loop now, skip else
    if test: continue          # Go to top of loop now
else:
    statements                 # If we didn't hit a 'break'
```

for loop

simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.py (3.4.4)

File Edit Format Run Options Window Help

```
for letter in 'Python':      # First Example
   print ('Current Letter :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:         # Second Example
   print ('Current fruit :', fruit)

print ("Good bye!")
```

Python 3.4.4 Shell

File Edit Shell Debug Options Window Help

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 2
tel)] on win32
Type "copyright", "credits" or "license(
>>>
 RESTART: C:/Documents and Settings/admi
e.py
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

for loop

*simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/s

File Edit Format Run Options Window Help

```
sum = 0
for x in [1, 2, 3, 4]:
      sum = sum + x

print(sum)

prod = 1
for item in [1, 2, 3, 4]:
    prod *= item

print(prod)

#>>>
#output:
    10
    24
```

prod = prod * item

x	sum
1	0
2	1
3	3
4	6
	10

item	prod
1	1
2	1
3	2
4	6
	24

$$\sum_{x=1}^{4} x \qquad \prod_{item=1}^{4} item$$

for loop

*simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.p

File Edit Format Run Options Window Help

```
for i in [2.3, [8, 9, 10], "city"]:
   print (i)

#>>>
#output:

    2.3
    [8, 9, 10]
    city
```

for loop

*simple.py - C:/Documents and Settings/admin/Desktop/intro-python/exa

File Edit Format Run Options Window Help

```
l = [4, 6, 7, 8, 10]
for i in l[2:]:
    print (i)


#>>>
#output:
     7
     8
     10
```

`range`

- **The `range` function specifies a range of integers:**
  - `range(`***start***`, `***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

- It can also accept a third value specifying the change between values.
  - `range(`***start***`, `***stop***`, `***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

- Example:

```
for x in range(1, 6):
    print (x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

range

- **The `range` function specifies a range of integers:**
  - `range(`***start***`, `***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

- It can also accept a third value specifying the change between values.
  - `range(`***start***`, `***stop***`, `***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

- Example:

```
for x in range(5, 0, -1):
    print (x)
print ("Blastoff!“)
```

Output:

```
5
4
3
2
1
Blastoff!
```

for loop

simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.py (3.4.4)

File Edit Format Run Options Window Help

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
   print ('Current fruit :', fruits[index])

print ("Good bye!")



#>>>
#output:

    Current fruit : banana
    Current fruit : apple
    Current fruit : mango
    Good bye!
```

range(3)
(0,1,2)

index	
0	banana
1	apple
2	mango

for loop

simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.py (3.4.4)

File Edit Format Run Options Window Help

```
S = 'spam'
for i in range(len(S)):        # For repeat counts 0..3
    S = S[1:] + S[:1]          # Move front item to end
    print(S, end=' ')


#>>>
#output:
    pams amsp mspa spam
```

4 → range(4) 0, 1, 2, 3

S[0]

i	S
0	spam
1	pams
2	amsp
3	mspa
	spam

for loop

simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.py (3.4.4)

File Edit Format Run Options Window Help

```
S = 'spam'
for i in range(len(S)):          # For positions 0..3
    X = S[i:] + S[:i]            # Rear part + front part
    print(X, end=' ')


#>>>
#output:
    spam pams amsp mspa
```

w → range(w) → 0, 1, 2, 3

i	X	S = 5
0	spam	spam
1	pams	
2	amsp	
3	mspa	

for loop

simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.py (3.4.4)

File Edit Format Run Options Window Help

```
L = [1, 2, 3]
for i in range(len(L)):
     X = L[i:] + L[:i]          # Works on any sequence type
     print(X, end=' ')


#>>>
#output:
    [1, 2, 3] [2, 3, 1] [3, 1, 2]
```

simple.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/simple.py (3.4.4)

File Edit Format Run Options Window Help

```
for num in range(10,20):   #to iterate between 10 to 20
    for i in range(2,num): #to iterate on the factors of the number
        if num%i == 0:     #to determine the first factor
            j=num/i        #to calculate the second factor
            print ('%d equals %d * %d' % (num,i,j))
            break          #to move to the next number, the #first FOR
    else:                  # else part of the loop
        print (num, 'is a prime number')


#>>>
#output:
    10 equals 2 * 5
    11 is a prime number
    12 equals 2 * 6
    13 is a prime number
    14 equals 2 * 7
    15 equals 3 * 5
    16 equals 2 * 8
    17 is a prime number
    18 equals 2 * 9
    19 is a prime number
```

num	i	j	nm	i	j
10	2	5	12	2	6
	3		13	2	
	4			3	
	5			4	
	9			5	
11	2			12	
	3				
	4				
	5				
	9				
	10				

Sum

sum.py - C:/Documents and Settings/admin/Desktop/intro-python/examples/17/sum.py (3.

File Edit Format Run Options Window Help

```
n=int(input("Enter the range: "))

sum=0
for i in range(n):
    sum=sum+i
print("The final sum  is: ", sum)

sumEven=0
for i in range(0, n, 2):
    sumEven=sumEven+i
print("The sum of evens is: ", sumEven)

sumOdd=0
for i in range(1, n, 2):
    sumOdd=sumOdd+i
print("The sum of odds is: ", sumOdd)
```

```
Enter the range: 50
The final sum  is:  1225
The sum of evens is:  600
The sum of odds is:  625
```

map function

- One of the common things we do with list and other sequences is applying an operation to each item and collect the result.
- The map() function applies a given function to each item of an iterable (list, tuple etc.) and returns a list of the results.
- The syntax of map() is:

```
map(function, iterable)
```

map function

*compa.py - /home/nowzari/Desktop/python/python-my/python/examples/15-for/compa.py (3.5

File Edit Format Run Options Window Help

```
res = list(map(int, '7239'))
print(res)

#output:

   [7, 2, 3, 9]
```

'7239' '7' 7
'2' 2
'3' 3
'9' 9
[7,2,3,9]

```
res = list(map(abs, [-8, -2, 3, -5]))
print(res)

#output:

   [8, 2, 3, 5]
```

[8, 2, 3, 5]

map function

compa.py - /home/nowzari/Desktop/python/python-my/python/examples/15-for/compa.py (3.5.2)

File Edit Format Run Options Window Help

```
L=['this' , 'is', 'a', 'class', 'test']
res = list(map(list, L))
print(res)

#output:

[['t', 'h', 'i', 's'], ['i', 's'], ['a'], ['c', 'l', 'a', 's', 's'], ['t', 'e', 's', 't']]
```

[['t', 'h', 'i', 's'],
['i', 's'],
['a'],
['c', 'l', 'a', 's', 's'],
['t', 'e', 's', 't']]

map function

*compa.py - /home/nowzari/Desktop/python/python-my/python/examples/1

File Edit Format Run Options Window Help

[1, 4, 9, 16, 25]

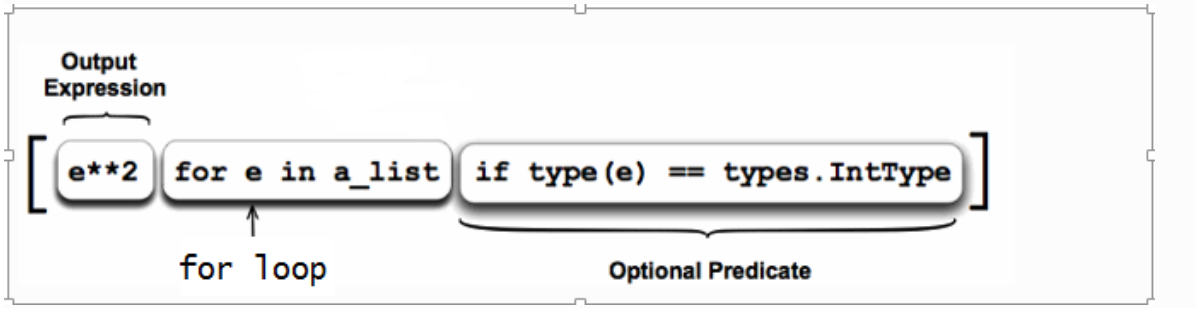
```
def sqr(x): return x ** 2

items = [1, 2, 3, 4, 5]
res=list(map(sqr, items))
print(res)

#output:

   [1, 4, 9, 16, 25]
```

Comprehensions

- In addition to sequence operations and list methods, Python includes a more advanced operation known as a *list comprehension expression, which turns out to be a powerful* way to process structures like list and array. This is called "list comprehensions"
- List comprehensions provide a concise way to create lists.
- It can be used to construct lists in a very natural, easy way, like a mathematician is used to do.

Comprehensions

- It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses. The expressions can be anything, meaning you can put in all kinds of objects in lists.



Comprehensions

- It can be used to construct lists in a very natural, easy way, like a mathematician is used to do.
- The following are common ways to describe lists (or sets, or tuples, or vectors) in mathematics.

```
S = {x² : x in {0 ... 9}}
V = (1, 2, 4, 8, ..., 2¹²)
M = {x | x in S and x even}
```

Comprehensions

```
S = {x² : x in {0 ... 9}}
V = (1, 2, 4, 8, ..., 2¹²)
M = {x | x in S and x even}
```

compa.py - /home/nowzari/Desktop/python/python-my/python/examples/15-for/compa.py (3.5.2)

File Edit Format Run Options Window Help

```
S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
M = [x for x in S if x % 2 == 0]
print ('S=', S)
print ('V=',V)
print ('M=',M)

#output:

   S= [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
   V= [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
   M= [0, 4, 16, 36, 64]
```

x	
0	0
1	4
4	16
9	
16	

x	x % 2
0	[0
1	
2	4
3	9]

Comprehensions

- Now, suppose we wish to collect the ASCII codes of *all characters in an entire string.* Perhaps the most straightforward approach is to use a simple for loop and append the results to a list:

```
res = []
for x in 'spam':
    res.append(ord(x))

print(res)

#output:
[115, 112, 97, 109]
```

x	res
	[]
's'	[115]
'p'	[115, 112]
'a'	[115, 112, 97]
'm'	[115, 112, 97, 109]

Comprehensions

- Now that we know about map, though, we can achieve similar results with a single function call without having to manage list construction in the code:

```
res = list(map(ord, 'spam'))
print(res)

#output:
[115, 112, 97, 109]
```

[115, 112, 97, 105]

Comprehensions

- However, we can get the same results from a list comprehension expression—while map maps a *function over an iterable, list comprehensions map an expression over a* sequence or other iterable:

```
res = [ord(x) for x in 'spam']
print(res)

#output:
[115, 112, 97, 109]
```

Comprehensions

- List comprehensions become more convenient, though, when we wish to apply an arbitrary expression to an iterable instead of a function:

```
res=[x ** 2 for x in range(10)]
print(res)

#output:

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Comprehensions

```
res = [x * y for x in [1, 2, 3] for y in [100, 200, 300]]
print(res)

#output:
[100, 200, 300, 200, 400, 600, 300, 600, 900]
```

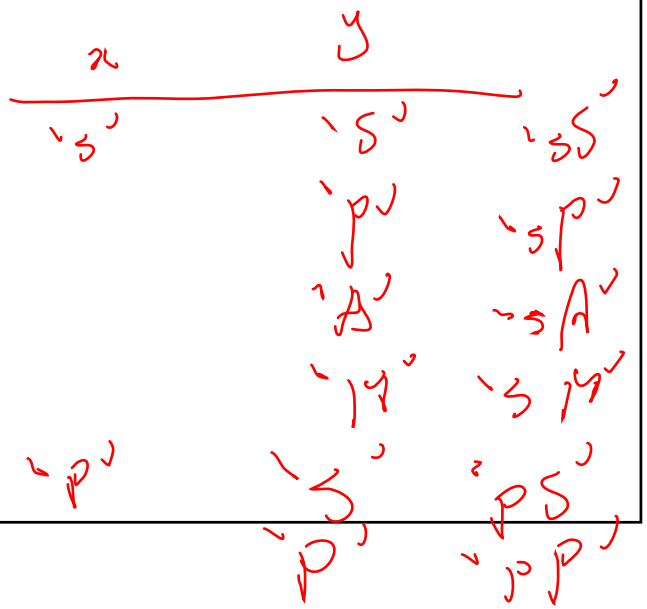
for x in [1, 2, 3]

for y in [100, 200, 300]

x	y	
1	100	[100,
	200	200,
	300	300,
2	100	200,
	200	400,
	300	600,
3	100	300,
	200	600,
	300	900]

Comprehensions

```
res=[x + y for x in 'spam' for y in 'SPAM']
print(res)

#output:

['sS', 'sP', 'sA', 'sM', 'pS', 'pP', 'pA', 'pM',
 'aS', 'aP', 'aA', 'aM', 'mS', 'mP', 'mA', 'mM']
```



Comprehensions

```
res=[x + y + z for x in 'spam' if x in 'sm'
               for y in 'SPAM' if y in ('P', 'A')
               for z in '123'  if z > '1']
print(res)

#output:

['sP2', 'sP3', 'sA2', 'sA3', 'mP2', 'mP3', 'mA2', 'mA3']
```

x	y	z	x	y	z
's'	'S'	'1'	's'	'P'	'2'
	'P'	'2'	's'	'P'	'3'
		'3'	's'	'A'	'2'
	'A'	'1'	's'	'A'	'3'
		'2'	'm'	'P'	'2'
		'3'	'm'	'P'	'3'
	'A'	'1'			
		'2'			
		'3'			
	'A'	'1'			
		'2'			
		'3'			
'p'		'1'			
'a'	'S'	'2'			
'm'	'P'	'3'			

Comprehensions

```
res=[[x, y] for x in range(5) if x % 2 == 0
            for y in range(5) if y % 2 == 1]
print(res)

#output:
```

```
[[0, 1], [0, 3], [2, 1], [2, 3], [4, 1], [4, 3]]
```

x	y	
0	0	[0, 1]
	1	[0, 3]
	2	[2, 1]
	3	[2, 2]
	4	
1		
2	0	
	1	
	2	
	3	

End