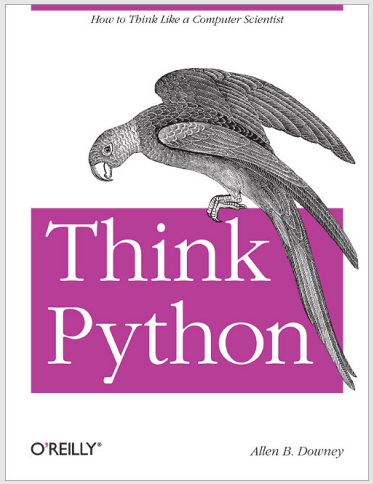
Contact

Dr. James Shackleford
shack@drexel.edu
Bossone 211

Office Hours: 3 – 4 pm (Tuesday)
Course Website: http://learn.dcollege.net

Textbook

Think Python
by Allen Downey
O'Reilly Press, 2015
ISBN-13: 978-1449330729
(Freely available in PDF format, check course website)



Grading

- 10% In-lab Programming Assignments
- 10% Take-Home Programming Assignments
- 35% Mid-term Exam
- 45% Final Exam

For Loops with List Indicies

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

For Loops with List Indicies

item

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

For Loops with List Indicies

item

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

For Loops with List Indicies

item

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

For Loops with List Indicies

item

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

For Loops with List Indicies

item

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

For Loops with List Indicies

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

len(mylist) = 5

For Loops with List Indicies

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

range(5) = [0, 1, 2, 3, 4]

For Loops with List Indicies

mylist[i]

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

[0, 1, 2, 3, 4]

i

For Loops with List Indicies

mylist[i]

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

[0, 1, 2, 3, 4]

i

For Loops with List Indicies

mylist[i]

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

[0, 1, 2, 3, 4]

i

For Loops with List Indicies

mylist[i]

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

[0, 1, 2, 3, 4]

i

For Loops with List Indicies

mylist[i]

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

[0, 1, 2, 3, 4]

i

For Loops with List Indicies

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    item = mylist[i]
    foo = item*10 + 4
    print foo
```

Functionally Equivalent

Both allow us the ***read*** items from a list and operate on them

For Loops with List Indicies

BUT

what about ***writing*** to the list **?**

(what if we want to modify the list items within the loop)

For Loops with List Indicies

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    foo = item*10 + 4
    print foo
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

Output:

```
?
```

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

Output:

```
[8, 23, 99, 4, 61]
```

why?

Non-“Pythonic” Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off bound to an element in mylist

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to an element in mylist

```
iterator = iter(mylist)
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to an element in mylist

```
iterator = iter(mylist)
item = iterator.next()
```

upon for-loop entry

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to an element in mylist

```
iterator = iter(mylist)
item = iterator.next()
```

upon each for-loop iteration

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to an element in mylist

```
iterator = iter(mylist)
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

create new int object

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to an element in mylist

```
iterator = iter(mylist)
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

rebind **`item`** **to new object**

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

Okay, so what is the "Pythonic" way of modifying a list in a loop?

`enumerate(mylist)` ***instead of*** `range(len(mylist))`

For Loops with List Indicies

Okay, so what is the "Pythonic" way of modifying a list in a loop?

`enumerate(mylist)` ***instead of*** `range(len(mylist))`

lets experiment for a sec...

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
>>> foo.next()
(4, 61)
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
>>> foo.next()
(4, 61)
>>> foo.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
>>> foo.next()
(4, 61)
>>> foo.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

okay, so its some sort of iterator

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
>>> foo.next()
(4, 61)
>>> foo.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

okay, so its some sort of iterator

but what is it returning?

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
>>> foo.next()
(4, 61)
>>> foo.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

these are tuples

A Very Brief Look at Tuples

What is a Tuple?

“An immutable list”

A Very Brief Look at Tuples

Defined Upon Creation (called “Packing”)

```
>>> test = (8, 23, 99, 4, 61)
>>> print test
(8, 23, 99, 4, 61)
```

A Very Brief Look at Tuples

Defined Upon Creation (called “Packing”)

```
>>> test = (8, 23, 99, 4, 61)
>>> print test
(8, 23, 99, 4, 61)
```

Parenthesis Optional (but customary)

```
>>> test = 8, 23, 99, 4, 61
>>> print test
(8, 23, 99, 4, 61)
```

A Very Brief Look at Tuples

Defined Upon Creation (called “Packing”)

```
>>> test = (8, 23, 99, 4, 61)
>>> print test
(8, 23, 99, 4, 61)
```

Parenthesis Optional (but customary)

```
>>> test = 8, 23, 99, 4, 61
>>> print test
(8, 23, 99, 4, 61)
```

Indexable

```
>>> test = (8, 23, 99, 4, 61)
>>> print test[2]
99
```

A Very Brief Look at Tuples

Immutable – Cannot Change!

```
>>> test = (8, 23, 99, 4, 61)
>>> test[2] = 327
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

A Very Brief Look at Tuples

Immutable – Cannot Change!

```
>>> test = (8, 23, 99, 4, 61)
>>> test[2] = 327
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Immutable – Cannot Append!

```
>>> test = (8, 23, 99, 4, 61)
>>> test.append(690)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
```

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
>>> name, sex, age, job = employee
```

unpacking

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
>>> name, sex, age, job = employee
>>> print name, sex, age, job
bob male 42 engineer
```

unpacking

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
>>> name, sex, age, job = employee
>>> print name, sex, age, job
bob male 42 engineer
```

unpacking

Non-"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> tmp = a
>>> a = b
>>> b = tmp
```

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
>>> name, sex, age, job = employee
>>> print name, sex, age, job
bob male 42 engineer
```

unpacking

Non-"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> tmp = a
>>> a = b
>>> b = tmp
```

"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> b, a = a, b
```

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
>>> name, sex, age, job = employee
>>> print name, sex, age, job
bob male 42 engineer
```

Non-"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> tmp = a
>>> a = b
>>> b = tmp
```

"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> b, a = a, b
```

packing

(creates an "anonymous" tuple object)

A Very Brief Look at Tuples

Tuples can be **Unpacked** as easily as they are **Packed**

```
>>> employee = ('bob', 'male', 42, 'engineer')
>>> name, sex, age, job = employee
>>> print name, sex, age, job
bob male 42 engineer
```

Non-"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> tmp = a
>>> a = b
>>> b = tmp
```

"Pythonic" Variable Swap

```
>>> a = 5
>>> b = 9

>>> b, a = a, b
```

unpacking "anonymous" tuple object into variables a and b

Enough Tuples – Back to...

`enumerate()`

For Loops with List Indicies

playing with enumerate()

```
>>> mylist = [8, 23, 99, 4, 61]
>>> foo = enumerate(mylist)
>>> foo
<enumerate object at 0x7f47c1b13d20>

>>> foo.next()
(0, 8)
>>> foo.next()
(1, 23)
>>> foo.next()
(2, 99)
>>> foo.next()
(3, 4)
>>> foo.next()
(4, 61)
>>> foo.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

okay, so its some sort of iterator

but what is it returning?

For Loops with List Indicies

`item` starts off **bound** to an element in mylist

```
iterator = iter(mylist)
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in mylist:
    item *= 10

print mylist
```

rebind **`item`** **to new object**

Output:

```
[8, 23, 99, 4, 61]
```

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in enumerate(mylist):
    item *= 10

print mylist
```

rebind **item to new object**

Output:

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

OMG!!/11!!

`enumerate( )`

RETURNS AN ITERATOR!

WHAT HAPPENS WHEN YOU CALL

`iter( )`

ON AN ITERATOR?

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23 ... 99, 4, 61]

for item in enu ... en(mylist)):
    item *= 10 ... 10

print mylist
```

Output:

```
... 0, 610]
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
>>> B is C
True
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
>>> B is C
True
>>> B.next()
(0, 5)
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
>>> B is C
True
>>> B.next()
(0, 5)
>>> C.next()
(1, 6)
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
>>> B is C
True
>>> B.next()
(0, 5)
>>> C.next()
(1, 6)
>>> C.next()
(2, 7)
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
>>> B is C
True
>>> B.next()
(0, 5)
>>> C.next()
(1, 6)
>>> C.next()
(2, 7)
>>> B.next()
(3, 8)
```

For Loops with List Indicies

`item` starts off **bound** to ??

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic

```
mylist = [8, 23

for item in enu
    item *= 10

print mylist
```

Output:

```
```

" Code

```
99, 4, 61]

en(mylist)):
10
```

```
0, 610]
```

LET'S FIND OUT

```
>>> A = (5, 6, 7, 8)
>>> B = enumerate(A)
>>> C = iter(B)
>>> B is C
True
>>> B.next()
(0, 5)
>>> C.next()
(1, 6)
>>> C.next()
(2, 7)
>>> B.next()
(3, 8)
```

YOU GET THE EXACT SAME ITERATOR BACK

For Loops with List Indicies

`item` starts off **bound** to the 1st tuple

```
iterator = iter(enumerate(mylist))
item = iterator.next()
```

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for item in enumerate(mylist):
    mylist[item[0]] *= 10

print mylist
```

Output:

Non-"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for i in range(len(mylist)):
    mylist[i] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

meh… binding **`item`** to a tuple isn't very elegant…
let's **UNPACK** in the for-loop

For Loops with List Indicies

index, item

```
iterator = iter(enumerate(mylist))
index, item = iterator.next()
```

UNPACKING

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for index, item in enumerate(mylist):
    mylist[index] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

For Loops with List Indicies

```
iterator = iter(enumerate(mylist))
index, item = iterator.next()
```

UNPACKING

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for index, item in enumerate(mylist):
    mylist[index] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

WE ARE NOT USING THE VARIABLE

`item`

it is common to use the variable name

`_`

when unpacking a tuple for elements we don't care about

For Loops with List Indicies

```
iterator = iter(enumerate(mylist))
index, item = iterator.next()
```

UNPACKING

"Pythonic" Code

```
mylist = [8, 23, 99, 4, 61]

for index, _ in enumerate(mylist):
    mylist[index] *= 10

print mylist
```

Output:

```
[80, 230, 990, 40, 610]
```

WE ARE NOT USING THE VARIABLE

`item`

it is common to use the variable name

`_`

when unpacking a tuple for elements we don't care about

Iterating Multiple Lists Simultaneously

What if I need to iterate through multiple lists at the same time?

`zip()`

Iterating Multiple Lists Simultaneously

What does `zip()` do?

```
>>> A = [22, 33, 44, 55]
>>> B = [10, 20, 30, 40]
>>> C = zip(A, B)
>>> print C
[(22, 10), (33, 20), (44, 30), (55, 40)]
```

It creates a list of tuples

Iterating Multiple Lists Simultaneously

This is a common design pattern:

```
A = [22, 33, 44, 55]
B = [10, 20, 30, 40]

for a, b in zip(A, B):
    print a, b, a+b
```

Output:

```
22 10 32
33 20 53
44 30 74
55 40 95
```

Iterating Multiple Lists Simultaneously

This is a common design pattern:

```
A = [22, 33, 44, 55]
B = [10, 20, 30, 40]

for a, b in zip(A, B):
    print a, b, a+b
```

zip(A, B) is a list

```
[ (22, 10),
  (33, 20),
  (44, 30),
  (55, 40)]
```

Output:

```
22 10 32
33 20 53
44 30 74
55 40 95
```

Iterating Multiple Lists Simultaneously

We can even use **`enumerate()`** with **`zip()`**

```
A = [22, 33, 44, 55]
B = [10, 20, 30, 40]

for index, (a, b) in enumerate(zip(A, B)):
    print index, a, b, a+b
```

Output:

```
0 22 10 32
1 33 20 53
2 44 30 74
3 55 40 95
```

iterator = enumerate(zip(A, B))

>>> iterator.next()
(0, (22, 10))

>>> iterator.next()
(1, (33, 20))

>>> iterator.next()
(2, (44, 30))

>>> iterator.next()
(3, (55, 40))