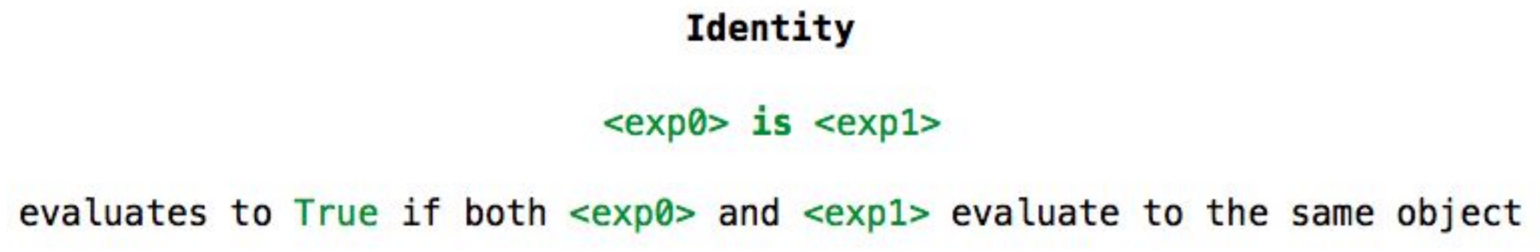
Lecture 8

List Comprehensions

Some slides were borrowed from CS88, Berkeley

Last time: Identity Operators: is   is not

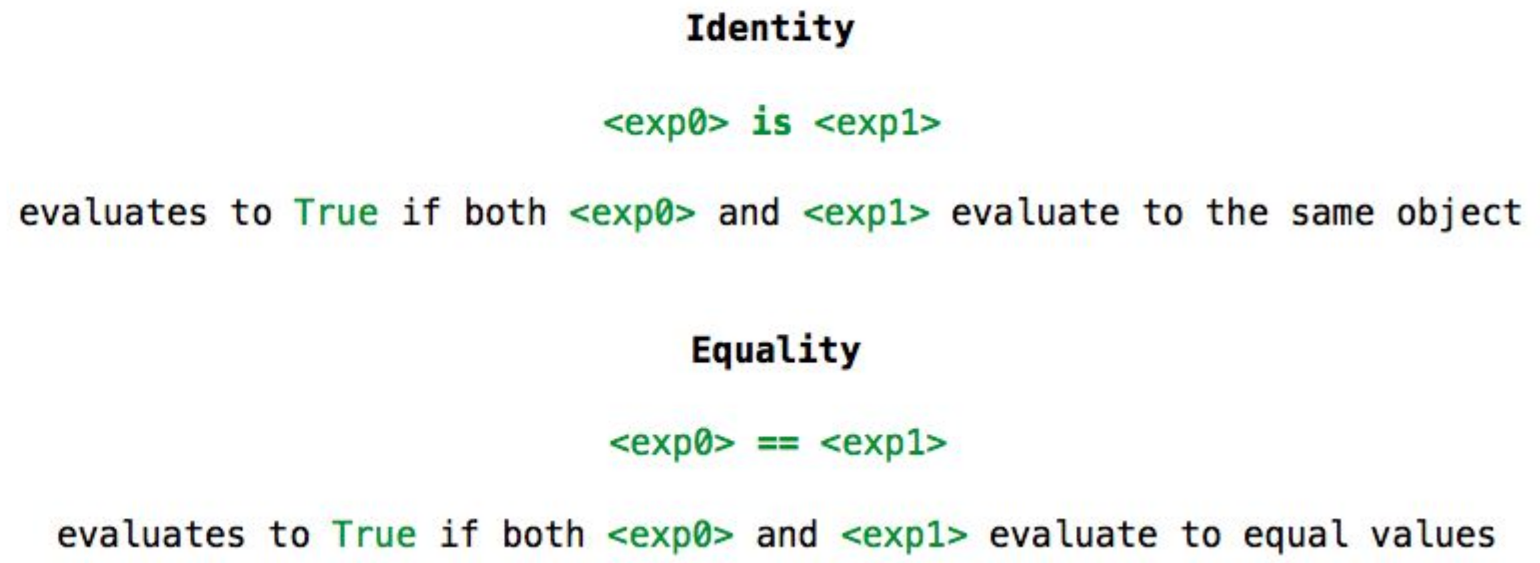


Identity

`<exp0> is <exp1>`

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Identity Operators



Identity

```
<exp0> is <exp1>
```

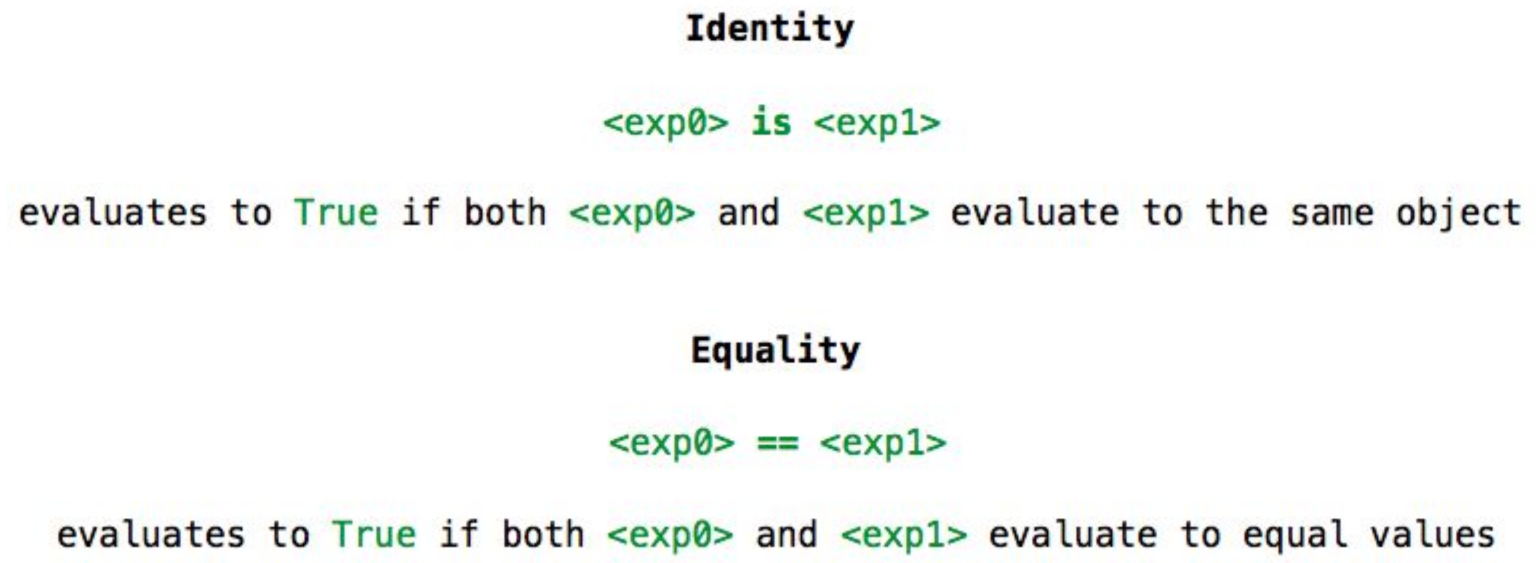
evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

```
<exp0> == <exp1>
```

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identity Operators



Identity

`<exp0> is <exp1>`

evaluates to True if both <exp0> and <exp1> evaluate to the same object

Equality

`<exp0> == <exp1>`

evaluates to True if both <exp0> and <exp1> evaluate to equal values

Identical objects are always equal values

Parameter passing: Output?

```
def test (x):
    x = x + 1
y = 10
test(y)
print(y)
```

A: 10

B: 11

C: None

D: Error

E: I do not know

Parameter passing: Output?

```
def test (x):
    x[0] = x[0] + 1
y = [1, 2, 3]
test(y)
print(y)
```

A: [1, 2, 3]

B: [2, 2, 3]

C: None

D: Error

E: I do not know

Parameter passing: Output?

```
def test (x):
    x[0] = x[0] + 1
y = (1, 2, 3)
test(y)
print(y)
```

A: (1, 2, 3)

B: (2, 2, 3)

C: None

D: Error

E: I do not know

What is the output?

```
var = ([1, 2], [3, 4])
copy = var
var[0][1] = "changed"
var = ([1, "changed"], [3, 4])
print(copy is var)
```

A : Error

B: None

C: ([1, “changed”], [3, 4])

D: True

E: False

What is the output?

```
def func(lst):
    new_list = lst
    new_list[0] = 'changed'

param = [1, 2, 3, 4, 5]
func(param)
print(param)
```

A : Error

B: None

C: [1, 2, 3, 4, 5]

D: [“changed”, 2, 3, 4, 5]

E: None of the above

What is the output?

```
def func(lst):
    new_list = lst
    new_list[0] = ‘m’

param = “string”
func(param)
print(param)
```

A : Error

B: None

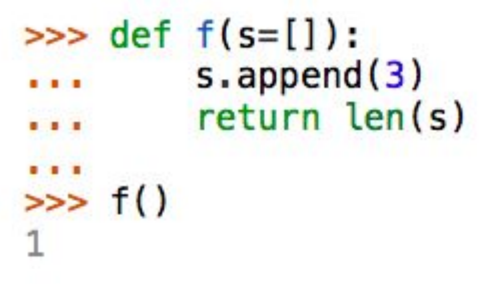
C: string

D: mtring

E: None of the above

Mutable Default Arguments are Dangerous

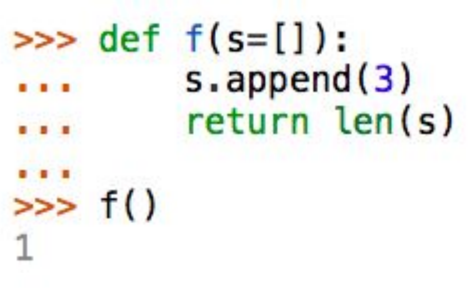
- A *default* argument value is part of a function value, not generated by a call



```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
```

Mutable Default Arguments are Dangerous

- A default argument value is part of a function value, not generated by a call

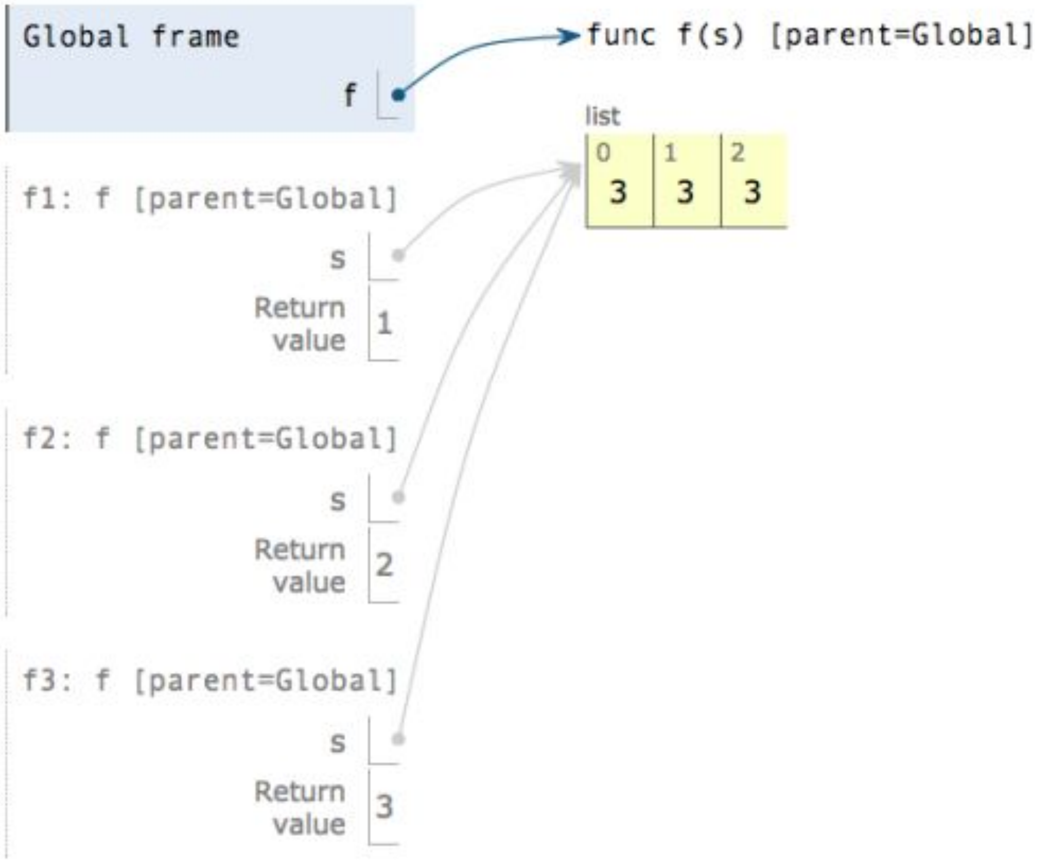


```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
```

What will happen if I call `f()` again?

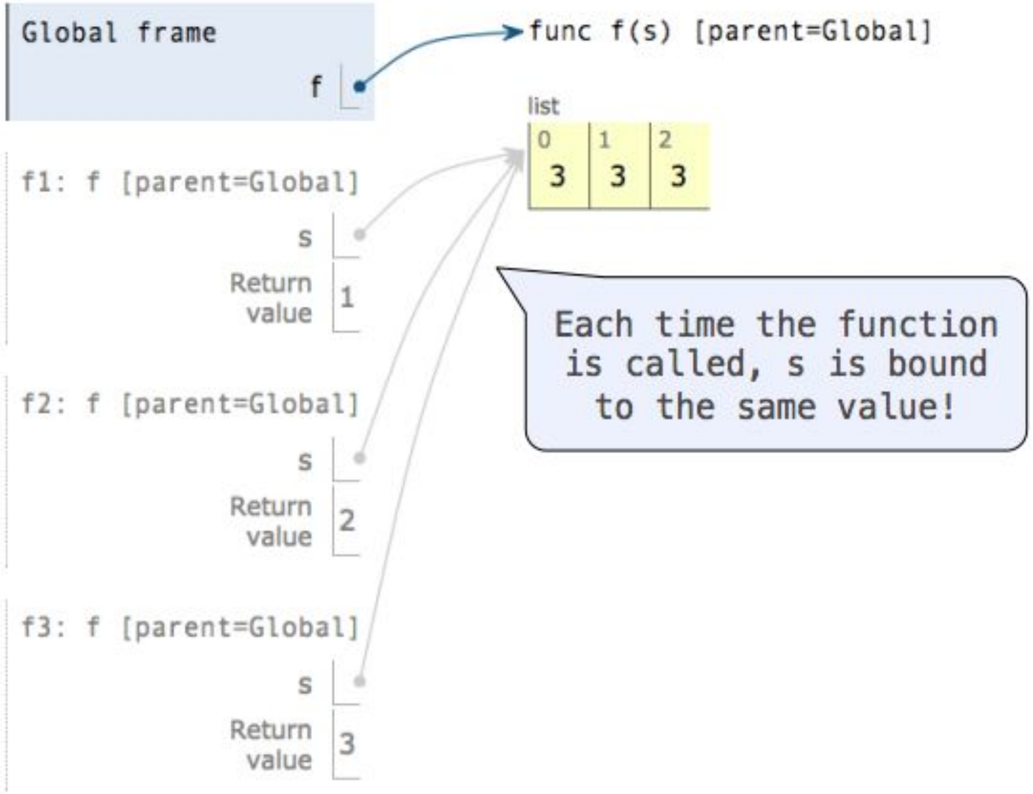
A: 1
B: 2
C: 3
D: 0
E: Error

```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```



A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```



Question from last class

How does Python store/represent a string?

- A string in Python is a **sequence** of characters.
  - Not a *list* of characters. Why?

- Every character is has its own integer representation
  - Binary to be more precise

- Internal representation: Strings are an array of integers

Using _

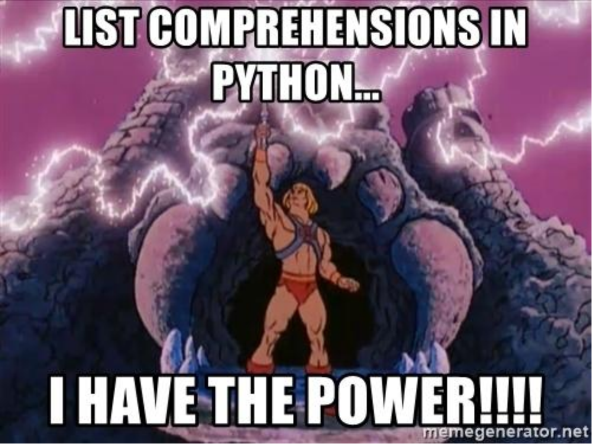
Use of ranges

```
def multiple_print(num):
    for _ in range(num):
        print(“midterm rocks!”)



multiple_print(10)
```

Use of ranges

```
def multiple_print(num):
    for _ in range(num):
        print("midterm rocks!")
```

When we do not care about the name since we are not using it.



List Comprehensions

Came from math

It is Python's way of implementing a well-known notation for sets as used by mathematicians.

- the square numbers of the natural numbers are, for example, created by

$$\{ x^2 \mid x \in \mathbb{N} \}$$

Came from math

it is Python's way of implementing a well-known notation for sets as used by mathematicians.

- the square numbers of the natural numbers are, for example, created by

$$\{ x^2 \mid x \in N \}$$

List comprehension is an elegant way to define and create list in Python.

Example

- `grades = [10, 40, 20, 30, 35]`
- `# Double them!`

- `doubled_grades = [20, 80, 40, 60, 70]`

- `What solution comes to mind?`

Example

- grades = [10, 40, 20, 30, 35]
- # Double them!
- dbl_grades = [20, 80, 40, 60, 70]
- What solution comes to mind? # I hope a map function :)

```
grades = [10, 40, 20, 30, 35]

dbl_grades=[]
for i in grades:
    dbl_grades.append(i * 2)
```

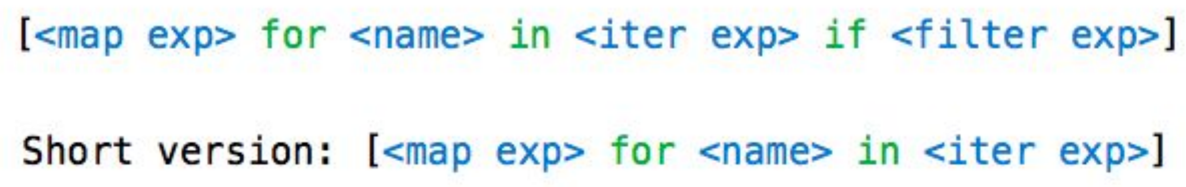
List Comprehension

- grades = [10, 40, 20, 30, 35]
- # Double them!
- dbl_grades = [20, 80, 40, 60, 70]

```
grades = [10, 40, 20, 30, 35]

dbl_grades=[]
for i in grades:
    dbl_grades.append(i * 2)
```

```
grades = [10, 40, 20, 30, 35]

dbl_grades=[i*2 for i in grades]
```

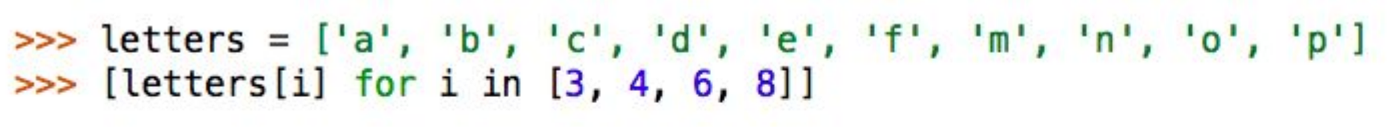
List Comprehensions



```
[<map exp> for <name> in <iter exp> if <filter exp>]

Short version: [<map exp> for <name> in <iter exp>]
```

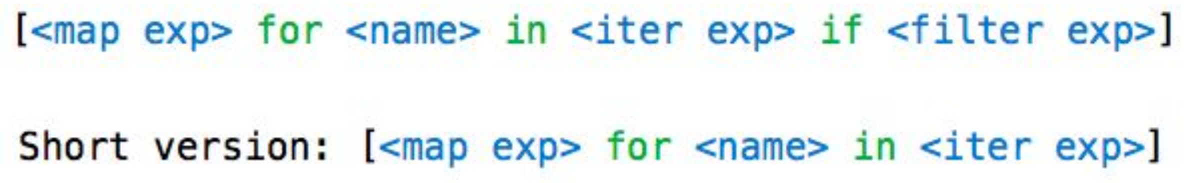
```
grades = [10, 40, 20, 30, 35]

dbl_grades=[i*2 for i in grades]

# Using Short version
```

List Comprehensions



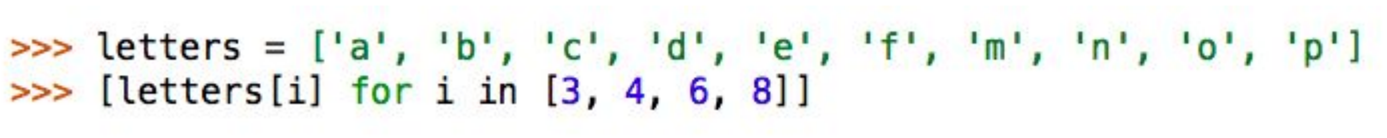
```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']
>>> [letters[i] for i in [3, 4, 6, 8]]
```

output?



```
[<map exp> for <name> in <iter exp> if <filter exp>]

Short version: [<map exp> for <name> in <iter exp>]
```

List Comprehensions



```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'm', 'n', 'o', 'p']
>>> [letters[i] for i in [3, 4, 6, 8]]
```

```
['d', 'e', 'm', 'o']
```

Problem. Square evens only

```
[<map exp> for <name> in <iter exp> if <filter exp>]
```

```
grades = [1,2,3,4,5,6,7,8,9,10]

sq_evens = []
for i in grades:
    if i % 2 == 0:
        sq_evens.append(i**2)
```

```
A: sq_evens = [i**2 for i in grades if i%2 == 0]
```

```
B: sq_evens = [i**2: for i in grades if i%2 == 0]
```

```
C: sq_evens = [i**2 for i in grades if i%2 != 0]
```

```
D: sq_evens = [for i in grades if i%2 == 0, i**2]
```

Once more

```
s1 = "abc"
s2 = "def"

lst = [[j + k] for j in s1 for k in s2]
```

A: ['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']

B: [['ad'], ['ae'], ['af'], ['bd'], ['be'], ['bf'], ['cd'], ['ce'], ['cf']]

C: [['ad'], ['ae'], ['af']]

D: ['ad', ‘ae’, ‘af’]

E: None of the above

Practice: convert C to F

```
ctemp  = [24, 20, 56, 32, 10]

ftemps = []

for c in ctemps:

    f = C_to_F(c)

    ftemps.append(f)
```

With a list comprehension:

Practice: convert C to F

```
ctemp  = [24, 20, 56, 32, 10]

ftemps = []

for c in ctemps:

    f = C_to_F(c)

    ftemps.append(f)
```

With a list comprehension:

```
ftemps  = [C_to_F(c) for c in ctemps]
```

Practice:

With a list comprehension: [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]

What is the code that outputs this list?

Practice:

With a list comprehension: [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]

```
locs = [(x, y) for x in range(3) for y in range(2)]
```

Practice:

With a list comprehension:

Create a list of integers which specify the length of each word in a certain sentence, but only if the word is not the word "the".

```
sentence = "What did the fish say when it swam into the wall? Dam!"
words = sentence.split()
```

Practice:

With a list comprehension:

Create a list of integers which specify the length of each word in a certain sentence, but only if the word is not the word "the".

```
sentence = "What did the fish say when it swam into the wall? Dam!"
words = sentence.split()

word_lengths = [len(word) for word in words if word != "the"]
```

Output?

```
[ [ 1 if i_idx == row_idx else 0 for i_idx in range(0, 3) ] for row_idx in range(0, 3) ]
```

Check point

```
[x for x in 'DATA SCIENCE' if x in ['A','E','I','O','U']]
```

A: ['D', 'T', ' ', 'S', 'C', 'N', 'C']

B: ['D', 'T', ' ', 'S', 'C', 'N',]

C: ['A', 'I', 'E']

D: ['A', 'A', 'I', 'E', 'E']

E: Something else