List Practice

Introduction to Computer Programming

Dr. Paul Vrbik

October 23, 2018

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()


>>>
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()


>>> xs.append([4,5,6])
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()


>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>>
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()


>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>> ys
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()


>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>> ys
[[1, 0, 1], [2, 1, 0], [3, 2, 1]]


>>>
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()

>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>> ys
[[1, 0, 1], [2, 1, 0], [3, 2, 1]]

>>> ys[0].extend([2, 3, 4])
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()

>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>> ys
[[1, 0, 1], [2, 1, 0], [3, 2, 1]]

>>> ys[0].extend([2, 3, 4])
[[1, 0, 1, 2, 3, 4], [2, 1, 0], [3, 2, 1]]
>>>
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()

>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>> ys
[[1, 0, 1], [2, 1, 0], [3, 2, 1]]

>>> ys[0].extend([2, 3, 4])
[[1, 0, 1, 2, 3, 4], [2, 1, 0], [3, 2, 1]]
>>> xs
```

Shallow-Copy

```
>>> xs = [[1, 0, 1], [2, 1, 0], [3, 2, 1]]
>>> ys = xs.copy()

>>> xs.append([4,5,6])
[[1, 0, 1], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
>>> ys
[[1, 0, 1], [2, 1, 0], [3, 2, 1]]

>>> ys[0].extend([2, 3, 4])
[[1, 0, 1, 2, 3, 4], [2, 1, 0], [3, 2, 1]]
>>> xs
[[1, 0, 1, 2, 3, 4], [2, 1, 0], [3, 2, 1], [4, 5, 6]]
```

Question

Students must be broken into teams so that no person is on two teams.

Assume students are uniquely identified by integers and write a function that checks if a list of teams satisfies the above condition.

```
def valid_teams(teams:List[List(int)]) -> bool:
```

List Nesting

In mathematics the Cartesian Product is used to generate pairs of points taken from two sets:

$$A \times B = \{(a, b) : a \in A \texttt{ and } b \in B\}.$$

For example $\mathbb{R}^2$ is the real-plane and

$$\{0,\, 1,\, 2\} \times \{a,\, b\} = \{(0, a),\, (1, a),\, (2, a),\, (0, b),\, (1, b),\, (2, b)\}.$$

Question

Implement `def cart_prod(A:list, B:list) -> list:`

Question

Write a function that constructs a list by "zig-zagging" two other lists:

```
def zig_zag(A:list, B:list) -> list:
    """
    >>> zig_zag([1, 2], [3, 4])
    [1, 3, 2, 4]
    >>> zig_zag([1, 2, 3], [4, 5])
    [1, 4, 2, 5, 3]
    >>> zig_zag([1, 2], [3, 4, 5])
    [1, 3, 2, 4, 5]
    """
```

Question

Suppose a positive integer is represented by a list of its digits as in:

$$234 \equiv [2, 3, 4]$$

Write a function that takes two such integers and return their sum using the list representation.

```
def list_add(A:List(int), B:List(int)) -> List(int):
    """
    >>> list_add([1, 2], [3, 4])
    [4, 6]
    >>> list_add([1, 8, 9], [4, 5])
    [2, 3, 4]
    """
```

Question (Advanced)

Write a function that "flattens" a list of lists.

```
def flatten(expression:list) -> list:
    """
    >>> flatten([])
    []
    >>> flatten([[1], [[2, 3], 4], [5]])
    [1, 2, 3, 4, 5]
    """
```

Next Class

1. Files,
2. Input, and
3. Output.