vector of length steps

```
delta = np.random.choice([-1,1], steps)
```

Which of the following snippets are equivalent to the above NumPy code (steps >= 0). All NumPy vectors should be Python lists.

scalar

A.
```
delta = random.choice([-1,1])
```

B.
```
delta = steps * random.choice([-1,1])
```

C.
```
delta = []
for i in range(2):
    delta.append(random.choice([-1,1]))
```

D.
```
delta = []
for i in range(steps):
    delta.append(random.choice([-1,1]))
```

Answer: D
The NumPy code creates a vector of length size of random integers that are either -1 or 1. So we want to create a list of length size of similar random integers using the random.choice function.

```
delta = np.random.choice([-1,1], steps)
walk = np.cumsum(delta)
```

Which of the following snippets are equivalent to the above NumPy code (steps is >= 0). NumPy vectors should be Python lists.

A.
```
pos = 0
walk = []
for i in range(steps):
  pos += random.choice([-1, 1])
  walk.append(pos)
```

B.
```
walk = []
for i in range(steps):
  delta=random.choice([-1,1])
  walk.append(delta)
```

C.
```
walk = []
for i in range(steps):
  if random.randint(0,1) == 0:
    walk.append(-1)
  else:
    walk.append(1)
```

D.
```
pos = 0
for i in range(steps):
  pos += random.choice([-1, 1])
```

Answer: A

delta is an array of randomly sample +1, -1, the cumsum function computes the cumulative sum, i.e., for [a, b, c] it computers [a, a+b, a+b+c]. Answer A implements the latter via the pos accumulator.

scalar

```
index = np.argmax(walk)
```

Which of the following snippets are equivalent to the above NumPy code?

A.
```
index = []
for i in range(1,len(a_list)):
  if a_list[i] > a_list[i-1]:
    index.append(i)
```

B.
```
index = []
for i in range(1,len(a_list)):
  if a_list[i] > a_list[index[-1]]:
    index.append(i)
```

C.
```
index = 0
for i in range(1,len(a_list)):
  if a_list[i] > a_list[i-1]:
    index = i
```

D.
```
index = 0
for i in range(1,len(a_list)):
  if a_list[i] > a_list[index]:
    index = i
```

Answer: D

We want to find the index of the largest value in the entire list. That will be a scalar, so A and B are incorrect. Answer C, would return the last index where the item was larger than its immediate predecessor (not the largest overall).

```
returns = np.cumprod(np.random.laplace(mean, scale, 240))
```

Which of the following snippets are equivalent to the above NumPy code? Assume there is a `laplace` function that has the mean and scale as arguments and returns a single sample.

A.
```
returns = []
for i in range(240):
  sample = laplace(mean, scale)
  returns.append(sample)
```

B.
```
returns = []
prod = 1.0
for i in range(240):
  sample = laplace(mean, scale)
  prod *= sample
  returns.append(prod)
```

C.
```
returns = []
prod = 1.0
for i in range(240):
  sample = laplace(mean, scale)
  returns.append(sample)
  prod *= sample
```

D.
```
returns = []
prod = 1.0
for i in range(240):
  sample = laplace(mean, scale)
  returns.append(prod)
  prod *= sample
```

Answer: B

The `cumprod` function computes a _cumulative_ product. Answers A and C only record the samples. Answer D has a "off by one", that is starts with the initial values and doesn't record the final product.