

STAT540: Computing in Statistics

Accessing Arrays, Boolean Masks, and Missing Values

Yen-Yi Ho

Department of Statistics, University of South Carolina

Section 1

Accessing Array Entries

Question 1

```
seq = np.arange(-1, 1, 1/4)    # 8 values: -1.00 ... 0.75  
seq[2:4]
```

How many values come back?

- A.** 3 — indices 2, 3 and 4
- B.** 2 — indices 2 and 3
- C.** 4 — indices 0 through 3
- D.** 6 — everything except indices 2 and 3

Question 1

```
seq = np.arange(-1, 1, 1/4)    # 8 values: -1.00 ... 0.75  
seq[2:4]
```

How many values come back?

- A. 3 — indices 2, 3 and 4
- B. 2 — indices 2 and 3
- C. 4 — indices 0 through 3
- D. 6 — everything except indices 2 and 3

Answer: B — you get [-0.5, -0.25].

A Python slice **excludes its stop value**. `seq[2:4]` is “from 2, up to but not including 4”. In R, `seq[2:4]` gives you *three* elements and starts counting at 1 — so the same expression returns a different set of numbers in each language, with no error either way.

Question 2

```
M = np.array([[1, 2, 3], [4, 5, 6]])  
even = M % 2 == 0  
M[even]
```

What is the shape of the result?

- A.** (3,) — a flat array [2 4 6]
- B.** (2, 3) — same shape, odd entries blanked
- C.** (2, 3) of True/False
- D.** (1, 3) — one row

Question 2

```
M = np.array([[1, 2, 3], [4, 5, 6]])  
even = M % 2 == 0  
M[even]
```

What is the shape of the result?

- A. (3,) — a flat array [2 4 6]
- B. (2, 3) — same shape, odd entries blanked
- C. (2, 3) of True/False
- D. (1, 3) — one row

Answer: A.

A boolean mask returns **only the selected entries**, flattened to 1-D. It cannot stay a matrix: the chosen entries need not form a rectangle. Here two come from row 0 and one from row 1, so [2 4 6] is the only shape available.

`M[~even]` flips the mask and gives the odd entries, [1 3 5].

Section 2

Masking and Assignment

Question 3

```
A = np.reshape(np.arange(24), (4, 6))  
A[A < 5] = 0
```

What happens to A?

- A. Nothing — `A[A < 5]` is a copy
- B. A shrinks to the 5 entries that were changed
- C. An error — you cannot assign through a mask
- D. A is modified **in place**; its first row starts
0 0 0 0 0 5

Question 3

```
A = np.reshape(np.arange(24), (4, 6))  
A[A < 5] = 0
```

What happens to A?

- A. Nothing — `A[A < 5]` is a copy
- B. A shrinks to the 5 entries that were changed
- C. An error — you cannot assign through a mask
- D. A is modified **in place**; its first row starts
0 0 0 0 0 5

Answer: D.

A mask on the **left** of `=` selects where to write. The array keeps its (4, 6) shape and the matching entries are overwritten — no copy, no new object, and the original values are gone.

Compare with Question 3: the same mask on the **right** of `=` *reads* and returns a flat copy. Same syntax, two different jobs.

Section 3

Missing Values

Question 4

```
D = rng.random((20, 3))  
D[0, 0] = None           # D is a float array  
np.max(D)
```

What does `np.max(D)` return?

- A.** The largest of the remaining values
- B.** None
- C.** nan
- D.** An error — cannot take a max with a missing value

Question 4

```
D = rng.random((20, 3))  
D[0, 0] = None           # D is a float array  
np.max(D)
```

What does `np.max(D)` return?

- A. The largest of the remaining values
- B. None
- C. nan
- D. An error — cannot take a max with a missing value

Answer: C.

None stored in a float array becomes **nan**, and **nan propagates** — one missing value makes the maximum, the mean and the sum all **nan**. Nothing warns you; you just get **nan** where a number should be.

Use the nan-aware versions: `np.nanmax`, `np.nanmean`, `np.nansum`. They are NumPy's answer to R's `na.rm = TRUE`.

Q	Idea	Answer
1	A slice excludes its stop value	B
2	A mask on the right <i>reads</i> , and flattens to 1-D	A
3	A mask on the left <i>writes</i> , in place	D
4	nan propagates — use <code>np.nanmax</code>	C

A mask reads on the right and writes on the left.