The following expression made a big splash on the internet

$$8 \div 2(2+2)$$

Which of the following is the equivalent Python code?

A. 8 // 2 * 4
B. 8 / (2 * 2 + 2)
C. 8 // (2 * 4)
D. None of the above

Answer: A
Python follows PEMDAS, or more specifically P E MD AS where MD and AS have equivalent priority, and “ties” are broken left to right. Note that 8 / 2 * 4 could also be valid depending on how strictly you interpret the above as an integer vs. floating point expression (i.e., is it 16 or 16.0). However, an even better answer is (8 // 2) * 4. Why? Because it is less confusing. While is A is not ambiguous, it can be confusing and we want to write code that is easy to understand. Style matters!

```
y = 3 + 2 + 1
y = y + 2
```

At the end of this code, what value is assigned to `y`?

A. 2
B. 6
C. 8
D. This code will cause an error

Answer: C
After the first line, y is assigned a value of 6. On the second line, y is first evaluated (to 6), before sum and then reassignment

```
y = 3
x = 7
y = 3 + 2 + 1
```

At the end of this code, what will the memory look like?

| | |
|---|---|
| A. y: 6 | B. y: 3<br>x: 7<br>y: 6 |
| C. y: 6<br>x: 7 | D. y: 3<br>x: 7 |

Answer: C
Both y and x are defined, but each variable, i.e., y, can only have one value at a time

```
x = 6
print(4x)
```

What would this code print?

A. 6
B. 24
C. 4x
D. Nothing, this code produces a syntax error

Answer: D
While in mathematical notation we can express multiplication by placing two operators next to each other, in Python (and most programming languages) we need to explicitly express multiplication with the * operator.