Worksheet: Understanding Operators Through Meaning and Guarantees

Do not run Python until you have reasoned through each problem. Answer in complete sentences. Focus on why the language behaves this way, not just what rules are followed.

Part I — Meaning Before Rules

1. Expression Without Parentheses

```
result = 12 - 4 * 2
```

a) What is the value?
b) Why does this expression have a single, agreed-upon meaning even without parentheses?
c) What would go wrong in a programming language if expressions like this did not have that property?

2. Explicit Grouping

```
result = (12 - 4) * 2
```

a) What changed compared to #1?
b) Why do parentheses change the meaning of an expression, not just its appearance?
c) Why is it important that parentheses always override default interpretation?

3. Grouping and Interpretation

```
result = 100 / (5 * 2 + 5)
```

a) Evaluate step by step.
b) Why is this not equivalent to (100 / 5) * 2 + 5?
c) What must be true so every reader interprets this expression the same way?

Part II — Division as Structure

4. Bounded Results

```
result = 37 % 8
```

a) What is the value?
b) Why is the result always constrained to a fixed range?
c) Why is that guarantee more important than the specific value?

5. Quotient and Offset

```
(37 // 8, 37 % 8)
```

a) What does each value represent?
b) Why do these two values together preserve more information than either alone?
c) Why does discarding one cause information loss?

Part III — Indexing and Reversibility

6. Linear Index to Grid

```
A grid has 6 columns. Let i = 29.
row = i // 6
col = i % 6
```

a) What are row and col?
b) Why will col never be outside the grid?
c) What guarantee does this mapping provide that makes it safe to use?

7. Grid to Linear Index

```
row = 4
col = 1
cols = 6
index = row * cols + col
```

a) What is the index?
b) Why does this undo the previous mapping?
c) Why is reversibility important in programs?

Part IV — Exponentiation and Grouping

8. No Parentheses

```
result = 2 ** 3 ** 2
```

a) Rewrite with parentheses to show Python's interpretation.
b) What is the value?
c) Why must Python choose one interpretation instead of leaving it ambiguous?

9. Explicit Grouping

```
result = (2 ** 3) ** 2
```

a) What changed compared to #8?
b) Why does changing grouping change the result?
c) Why must grouping rules be predictable?

10. Subtle Case

```
result = 9 ** 1 ** 2
```

a) What is the value?
b) Why is this not 81?
c) What consistency guarantee is being enforced?

Part V — Information Loss and Reconstruction

11. Partial Breakdown

```
result = 53 // 10 + 53 % 10
```

a) What information is lost?
b) Why can't the original number be recovered?
c) When is information loss acceptable?

12. Reconstruction

```
result = (n // 10) * 10 + (n % 10)
```

a) Why does this work only under certain assumptions?
b) What assumption is required?
c) When would this fail?

Part VI — Rounding and Representation

13. Rounding to Integer

```
result = round(7 / 3)
```

a) Why is the result an integer?
b) What decision is Python making for you?
c) Why can this be dangerous?

14. Controlling Precision

```
result = round(7 / 3, 2)
```

a) What does the 2 control?
b) What stays the same as precision changes?
c) What changes?

15. Same Value, Different Meaning

```
a = round(7 / 3)
b = round(7 / 3, 0)
```

a) Why do these look the same when printed?
b) Why are they not the same?
c) Why does this difference matter later?

Part VII — Synthesis

16. Final Question
Choose one operator from this worksheet and answer:
a) Why does this operator exist?
b) What guarantee does it provide?
c) When would using it violate your assumptions?