Game Programming with Python

Variables

Marcus Birkenkrahe

September 24, 2024

Contents

1 README

- The original notebook is written in Emacs Org-mode.
- An IPython notebook version is available on Colab.

2 Manipulating integer values

We'll start by learning how to manipulate numbers ('arithmetic').

- In the code block, execute the operation `2 + 2` (`CTRL + ENTER`).

```
2 + 2
```

- In the next code block, write `2 + 2` on one line, and `2 - 2` on the next line, then execute the block:

```
2 + 2
2 - 2
```

- [In Colab] Where did the first result go? Answer: you need to use `print` for every expression that you want to print out, otherwise only the last one evaluated will be shown.

3 Using operators, floats, integers

- You can add, subtract, multiply and divide in Python.

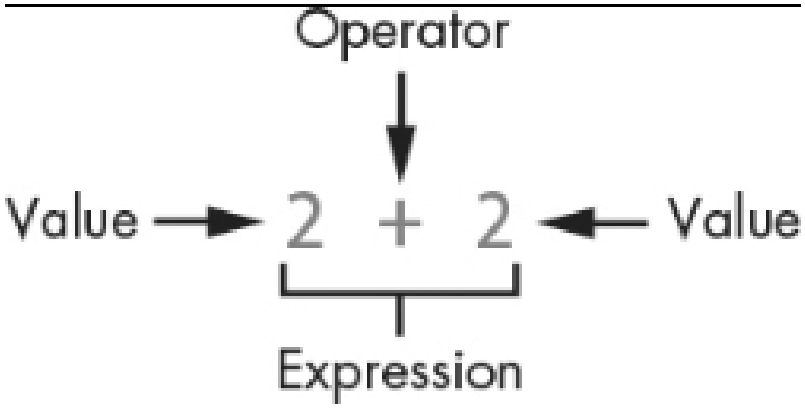
```
print(1e3 * 1e-3)
print(1/1)
```

```
1.0
1.0
```

- The first line above uses scientific notation for large numbers: `1e3` is `10 * 10 * 10 = 1000`, and `1e-3` is `1 / (10 * 10 * 10) = 1/1000` or `0.001`.

- Both operations result in a decimal (or floating-point) number (`1.0`), or *float*, rather than a whole (or integer) number (`1`).

- A number like `1` or `2.0` is a *value*. A math problem like `2 + 2` is an *expression*. Expressions are made up of values connected by operators (`+`).



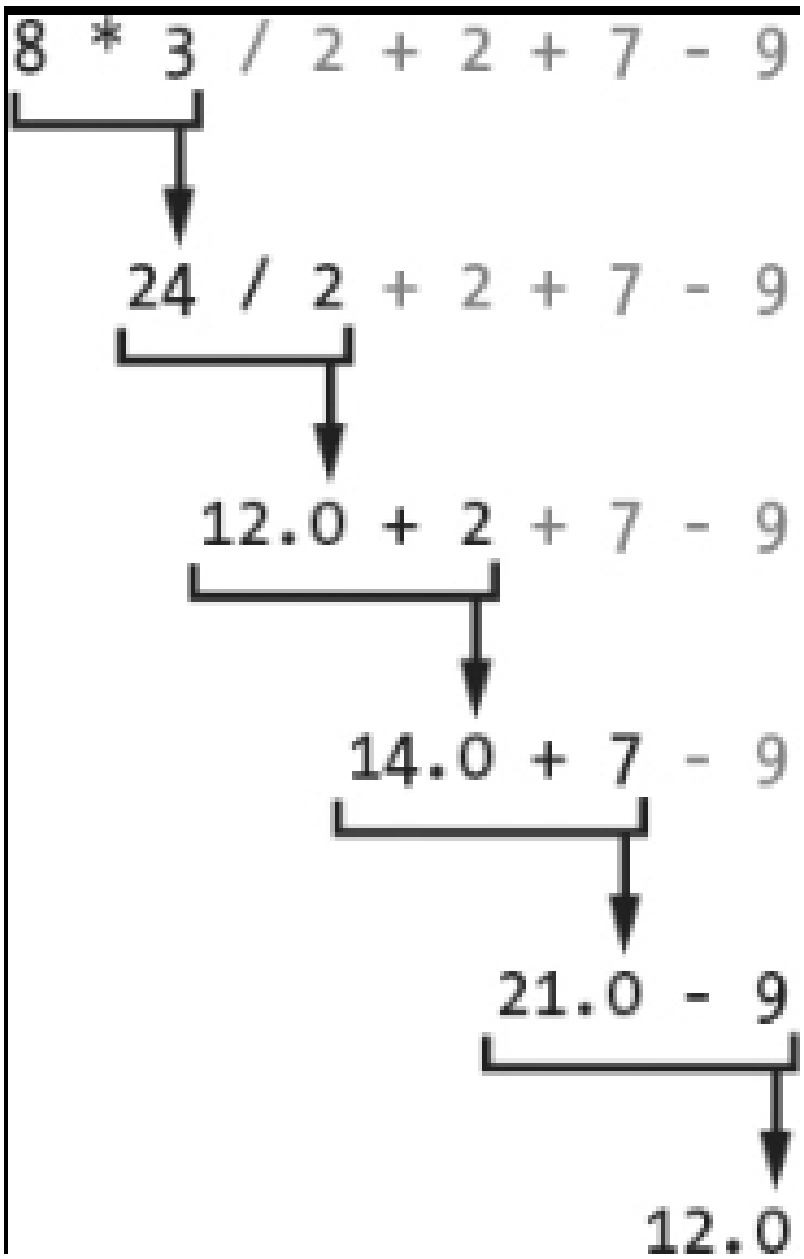
- The computer is obsessed with evaluating expressions. In the next code block, evaluate some expressions and `print` the results:

    1. `2 + 2 + 2 + 2 + 2`
    2. `8*6`
    3. `10 - 5 + 6`
    4. `2 + 2`

```
print(2 + 2 + 2 + 2 + 2)
print(8*6)
print(10 - 5 + 6)
print(2 +          2)
```

```
10
48
11
4
```

- When an expression is evaluated, Python has to observe an order of operations ("P+E+MD+AS"). The expression is always evaluated to a single value:



- Run the first line of the code in a code block and `print` the result:

```
print(8 * 3 / 2 + 2 + 7 - 9)
```

```
12.0
```

- Test question: Will the following expressions give the same or different results?

```
print(8 * 3 / 2 + 2 + 7 - 9)
print(2 + 7 - 9 + 8 * 3 / 2)
print(2 + 7 - 9 + 8 * (3 / 2))
print(2 + 8 * (3 / 2) + 7 - 9)
```

```
12.0
12.0
12.0
12.0
```

4 Making syntax errors

- Entering ~ 5 + ~ generates a `SyntaxError` because the `+` operator is *binary* and requires two arguments on either side:

```
5 +
```

- *Syntax errors* result from not observing the rules of the language - it's as if Yoda was saying "Home I go". This violates the SPO rule of English syntax - Subject + Predicate + Object.
- The difference between humans and machines: we can often, the computer can never recover from syntax errors.

5 Storing values in variables

- A variable is like a box that can hold a value.
- In the next code block, store the integer number `15` in a variable called `spam`:

```
spam = 15
```

- You've just written a *statement* or more specifically an *assignment statement* using the assignment operator `=`. There's no output until you ask for the value stored in `spam`.

```
print(spam)
```

```
15
```

- Python is case-sensitive, i.e. `SPAM` is different from `spam` or from `Spam`. You can test that by printing all of these:

```
print(spam)
print(SPAM)
print(Spam)
```

```
15
```

- The last two attempts result in a `NameError` because these variables were `not defined`, i.e. they were never assigned values.

6 Computing with variables

- Once a variable is defined, you can use it to compute. In the next code block, `print` the expressions `spam + 5` and `spam * spam`:

```
print(spam + 5)
print(spam * spam)
```

```
20
225
```

- In fact, you don't need two lines for this: put both expressions in the same `print` command:

```
print(spam + 5, spam * spam)
```

```
20 225
```

- Now change the value of `spam` to `3` and print the expressions again:

```
spam = 3
print(spam + 5, spam * spam)
```

```
8 9
```

- Do you think it's possible to do all of that in the `print` command, like this:

```
print(spam = 3, spam + 5, spam * spam)
```

- You encounter a third kind of error, a `TypeError`: inside `print`, `spam` is not recognized as part of `spam = 3`.
- However, if you change the `=` in the last command to a `==`, the code works:

```
print(spam == 3, spam + 5, spam * spam)
```

```
True 8 9
```

- This is because now you're printing a *value* as required by Python, the value is `True` because `spam` is actually equal to `3`. The `==` is a relational operator. It tests the equality of its left and its right hand operand.
- In the next code block, first alter the value of `spam` by adding `2` to itself like this: `spam = spam + 2`. In the following line, repeat the previous `print` command:

```
spam = spam + 2
print(spam == 3, spam + 5, spam * spam)
```

```
False 10 25
```

- Now, `spam == 3` is `False`, because the new value is $3 + 2 = 5$.
- In the next code block, define two more variables, `bacon` with the value `10`, and `eggs` with the value `15`.

```
bacon = 10
eggs = 15
```

- Enter `spam = bacon + eggs` in the next code block, then check the value of `spam`:

```
spam = bacon + eggs
print(spam)
```

```
25
```

7 Summary

- Expressions are values like `2` or `5.0` combined with operators like `+` or `/`.
- Expressions are evaluated and reduced to a single value.
- Values can be stored in variables to be remembered and used later.
- Python errors include `SyntaxError`, `TypeError` and `NameError`.