PYTHON BASICS

CSC 109 - Introduction to programming in Python - Summer 2023

Marcus Birkenkrahe

May 26, 2023

Contents

1 Python Basics

- Python is a rich high-level programming language (like C or R) with many features. To master it takes a long time (5-10 years).

- To write handy little programs that automate 'boring' tasks, you only need some basics:
  1. expressions 2 + 2
  2. data types integer
  3. variables spam
  4. statements spam = 1
  5. debugging dealing with errors
- When I lecture, you should always keep Python open to code along:
  1. Google Colab notebook
  2. IDLE interactive shell
  3. `python` on the command line
  4. Console in replit.com or DataCamp workspace
- The code is available as GitHub gist and in the `ipynb` directory.

2 Expressions: values and operators (gist)

- Open an interactive Python shell. I have changed the default settings in Colab to open with a "scratchpad" (not saved!).
- Enter the classic formula `2 + 2` at the prompt and press `RET` (Enter) to (hopefully) get the classic answer `4`.
- In Colab, if you run your code with `SHIFT + ENTER`, you get a new code cell right away. If you us `CTRL + ENTER` you get nothing but now you can add a text cell below with `CTRL + ALT + t`
- `2 + 2` is called an *expression*, a basic programming instruction.
- An expression consist of *values* (such as `2`) in computer memory, and *operators* (such as the binary operator `+`), which are *functions*.
- Expressions can always *evaluate* i.e. reduce to a single value - so you can e.g. use `2+2` anywhere instead of `4` because you know it's going to be reduced to `4`.
- Examples:
  1. use `2+2` as the *argument* of a `print` function.

2. use `2+2` as the argument of a `str` function.

- A single value like `2` is also an expression (it doesn't express anything else but itself) and evaluates to itself.

3 Error messages

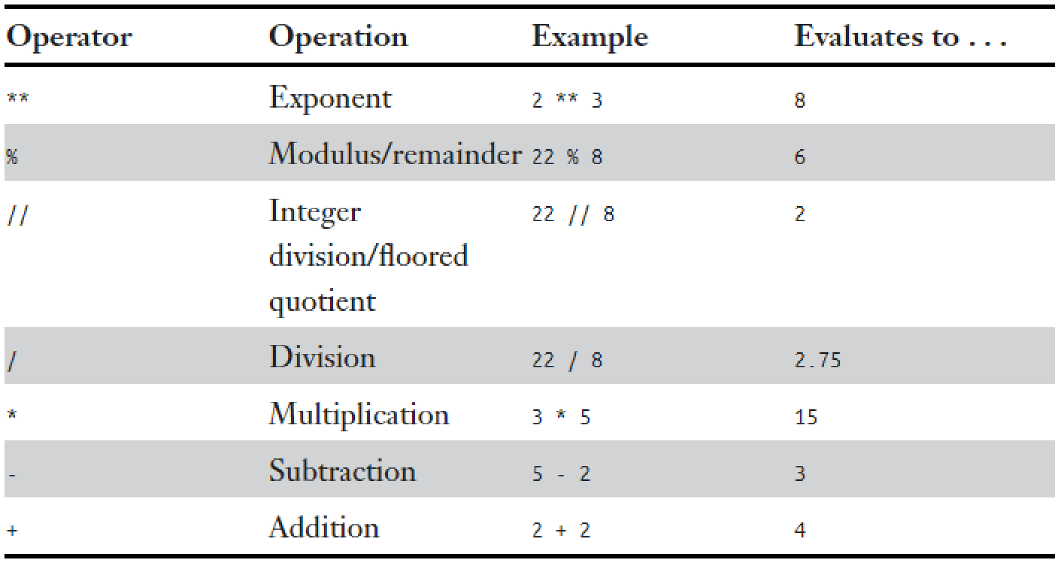
- When Python cannot evaluate an expression, it "throws" an error. Here is list of common error messages in Python with a plain English explanation (Sweigart, 2019).

- Let's create a couple of error messages using wrong expressions:

  1. Enter `2 +`
  2. Enter `2 + '2'`
  3. Enter `2` and then on the next line enter `2` again in the 2nd column
  4. Enter `2 + ++ 2` then change the first `+` to a `-`

4 Operators

- The table shows a lit of all math operators in Python. They are listed from highest to lowest precedence:



Operator	Operation	Example	Evaluates to . . .
`**`	Exponent	`2 ** 3`	`8`
`%`	Modulus/remainder	`22 % 8`	`6`
`//`	Integer division/floored quotient	`22 // 8`	`2`
`/`	Division	`22 / 8`	`2.75`
`*`	Multiplication	`3 * 5`	`15`
`-`	Subtraction	`5 - 2`	`3`
`+`	Addition	`2 + 2`	`4`

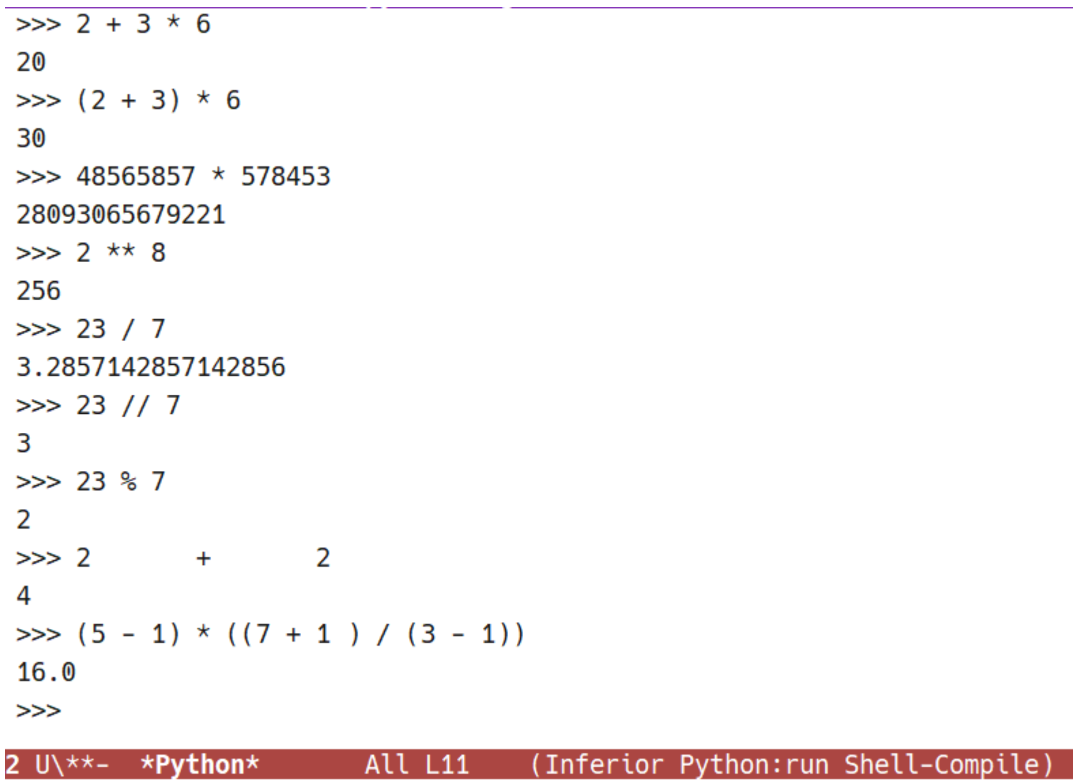
- The precedence is the order of operations: when Python gets an expression with more than one operator, it evaluates from left to right (you can force execution with parentheses).
- For example, the expression `-2+24/8` is evaluated as `1` and not as `2.75` because `(24/8)=3` and `3-2=1`:
    1. Enter `-2 + 24 / 8`
    2. Enter `(-2 + 24) / 8`
- So-called "whitespace" (empty space) between symbols does not matter, so `24/8` is evaluated identically to `24 / 8`.
- Enter the following expressions into the interactive shell:

```
2 + 3 * 6
(2 + 3) * 6
48565857 * 578453
2 ** 8
23 / 7
23 // 7
2      +     2
(5 - 1) * ((7 + 1 ) / (3 - 1))
```

- You should get this result:
- The next diagram shows how Python ruthlessly evaluates parts of the expression until it has reached a single value:

5 Variables

- A data type is a category for values: every value belongs to exactly one data type.
- Variables in Python do not need to be declared but they are dynamically typed, i.e. at runtime.
- Common data types are listed in this table:
- Python's names for these data types are: `int`, `float` and `str`.
- The `type` function reveals a value's or a variable's data type:



```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565857 * 578453
28093065679221
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2       +       2
4
>>> (5 - 1) * ((7 + 1 ) / (3 - 1))
16.0
>>>

2 U\**-  *Python*        All L11    (Inferior Python:run Shell-Compile)
```

Figure 1: Expressions in the interactive Python shell (in Emacs)

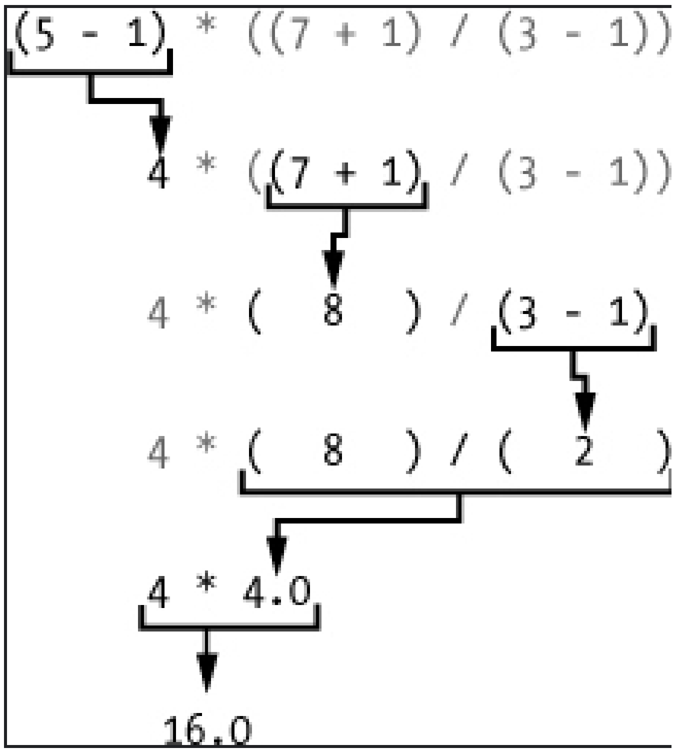
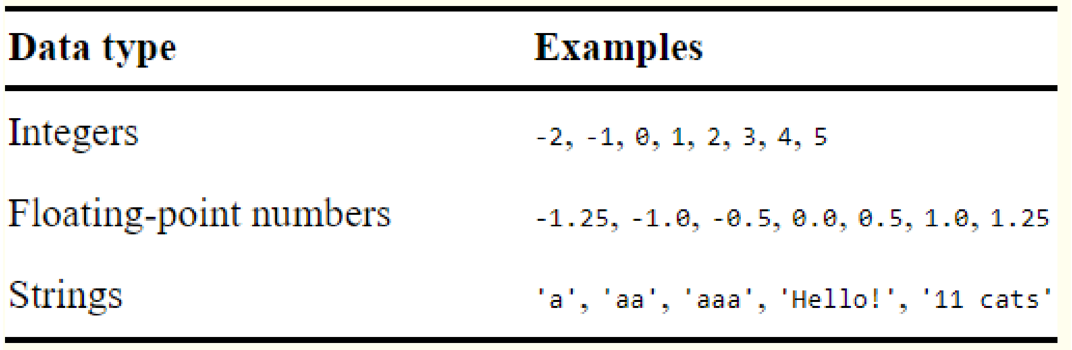


Figure 2: Evaluation of composite expression to a single value



Data type	Examples
Integers	-2, -1, 0, 1, 2, 3, 4, 5
Floating-point numbers	-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25
Strings	'a', 'aa', 'aaa', 'Hello!', '11 cats'

Figure 3: Common data types (Source: Sweigart, 2019)

```
type(-2)
type(2)
type(1.25)
type('a')
type('name')
type(a)
```

- Why does `type(a)` give a "Name Error"? Because Python expects a variable named `a`.

6 String concatenation and replication

- The meaning of an operator may change based on the data types of its operands.
- Enter the following examples in separate code cells (otherwise you only get the last result - or you have to add `print`).
- Examples:
  1. `'Alice' + 'Bob'`
  2. `'Alice' + 42`
- Python can only concatenate numbers or strings. You have to explicitly convert the 2nd argument to a string:
  1. `'Alice' + str(42)`
  2. `'Alice' + str(Bob)`
- Unless `Bob` is initialized as an integer, this will not work:
  1. `Bob = 42`
  2. `'Alice' + str(Bob)`
- The `*` operator can be used with one string and one integer value for replication:
  1. `'Alice' * 'Bob'`
  2. `'Alice' * 5.0`
  3. `'Alice' * 5`
  4. `'Alice' * int(5.0)`

7 Assignments: storing values in variables

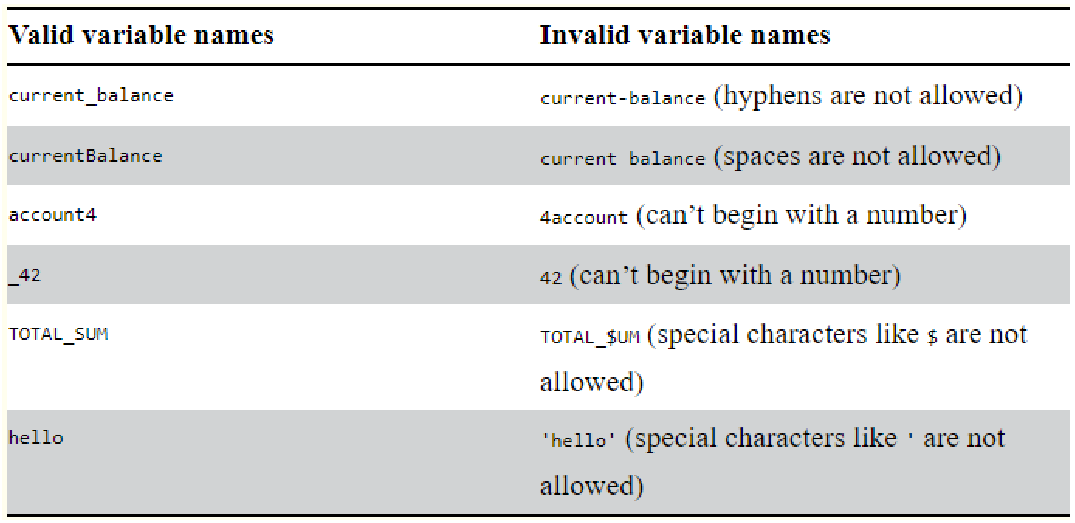
- A *variable* is like a box in the computer's memory where you can store a single value.
- You store values in variables with an `assignment statement`, consisting of: a variable name, the `=` operator, and the value.
- A variable is initialized or created the first time a value is stored in it.
- When a variable is assigned a new value, the old value is forgotten.
- To visualize this, open `pythontutor.com` and enter this code:

```
spam = 40
eggs = 2
spam + eggs
spam + eggs + spam
spam = spam + eggs
print(spam)
```

- Similarly for strings:

```
spam = 'Hello'
print(spam)
spam = 'Goodbye'
print(spam)
```

8 Variable names



Valid variable names	Invalid variable names
`current_balance`	`current-balance` (hyphens are not allowed)
`currentBalance`	`current balance` (spaces are not allowed)
`account4`	`4account` (can't begin with a number)
`_42`	`42` (can't begin with a number)
`TOTAL_SUM`	`TOTAL_$UM` (special characters like `$` are not allowed)
`hello`	`'hello'` (special characters like `'` are not allowed)

- You can name a variable anything as long as it obeys these rules:
  1. It can be only one word with no spaces
  2. It can only use letters, numbers and the underscore character (`_`)
  3. It can't begin with a number
- You should not use Python keywords, symbols, function or module names as your variables (though you may be allowed to).
- Variables in Python are case-sensitive.
- Some people prefer camel-case for variable names instead of underscores: `helloWorld` instead of `hello_world`. Either is OK.

9 TODO Back to 'hello world'

10 TODO Summary

- An instruction that evaluates to a single value is an **expression**. An instruction that doesn't is a **statement**.
- Data types are: integer (`int`), floating-point (`float`), string (`str`)

- Strings hold text and begin and end with quotes: `'Hello world!'`
- Strings can be concatenated (`+`) and replicated (`*`)
- Values can be stored in variables: `spam = 42`
- Variables can be used anywhere where values can be used in expressions: `spam + 1`
- Variable names: one word, letters, numbers (not at beginning), underscore only

11 TODO Glossary

TERM/COMMAND	MEANING

12 References

- Sweigart, A. (2016). Invent your own computer games with Python. NoStarch. URL: inventwithpython.com.
- Sweigart, A. (2019). Automate the boring stuff with Python. NoStarch. URL: automatetheboringstuff.com.