PYTHON BASICS

CSC 109 - Introduction to programming in Python - Summer 2023

Marcus Birkenkrahe

May 30, 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))
```

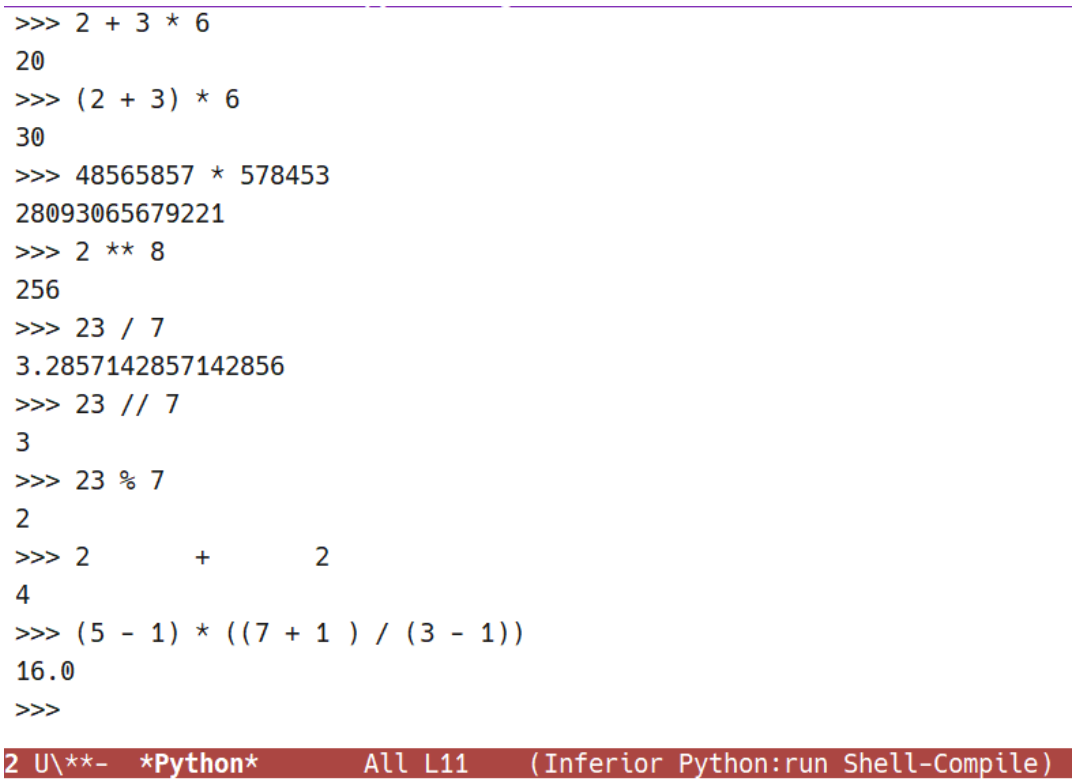

Figure 1: Expressions in the interactive Python shell (in Emacs)

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

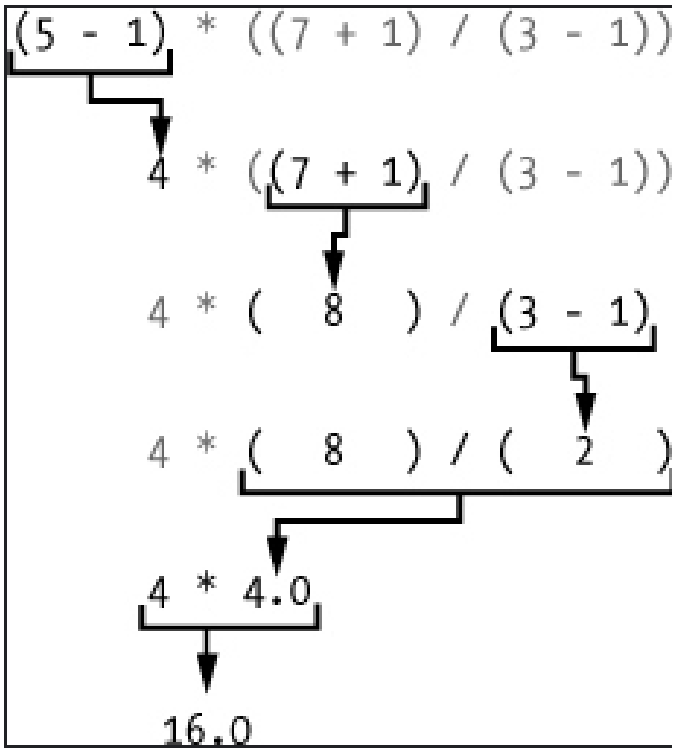

Figure 2: Evaluation of composite expression to a single value

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`.

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)

- The `type` function reveals a value's or a variable's data type:

```
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 Warming up: spooky season

- Problem: print "spooky" with 2 to 20 vowels (solution).
- **Let's do it together** - open a new Colab notebook `spooky.ipynb` for:
  1. solution flow (from input to output)
  2. variables (storing values)

Figure 4: "spooky" by Tony Coates (flickr.com)

3. functions and operators (doing stuff)
4. implementation (coding)
5. testing (debugging)
6. production (submission)

10 Understanding standard data streams

- We want to write a program that
  1. Says 'Hello world!'
  2. Asks for your name
  3. Greets you with your name
  4. Tells you how many characters your name has
  5. Asks for your age
  6. Tells you how old you're going to be in one year
- We're going to use this command sequence to learn a few functions useful to get input from the keyboard and manipulate text.
- Check the `help` for `input` in the Python reference manual, or in Colab, enter `input?` to get the *docstring*:

```
>>> help(input)
Help on built-in function input in module builtins:

input(prompt=None, /)
    Read a string from standard input.  The trailing newline is stripped.

    The prompt string, if given, is printed to standard output without a
    trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.
    On *nix systems, readline is used if available.
```

Figure 5: Python help for keyboard input() function

- What does this mean?
  1. `input` reads a string from the keyboard or from a file (*stdin*)
  2. If `input()` is used, the default `prompt` is missing (`None`)

3. If a prompt is used, it is printed without newline (*stdout*)
4. If CTRL-D (End Of File) is hit, an `EOFError` is raised.

- Standard input, output and error are the three data streams:

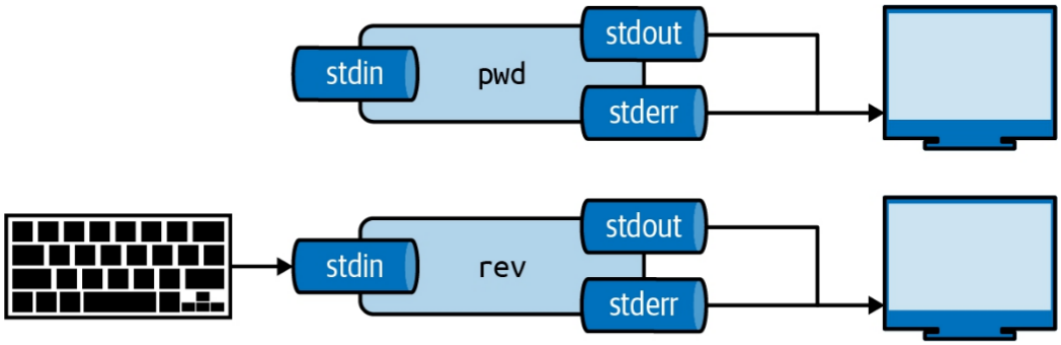

Figure 6: stdin, stdout, stderr for two shell commands

- Their standard direction is the screen but they can be redirected anywhere, e.g. into files:

```
rm hello 2&>/dev/null
echo "Hello, world" > hello
cat hello

Hello, world
```

11 Getting input from the keyboard

- Step 1: Ask for user's `name` and print out the number of characters in the name:

```
print('hello world')
print('What is your name?')
name = input()
print('Good to meet you,' + name)
print('Your name has', len(name), 'characters')
```

- Why did we not use the `+` operator in the last line? Try it.

- Step 2: ask for user's age and print out age one year from now:

```
print('What is your age?')
age = input()
print('You are going to be ' + str(int(age) + 1) + ' years old')
```

12 Python script infrastructure

- Not to forget about the Python script infrastructure:
  1. You can save the Python code of your notebook as `.py` file
  2. Open `File > Download > Download as .py`
  3. Look at the file (`""" ... """` is a multi-line comment)
  4. You can run the script on the terminal

13 Getting keyboard input with a prompt

- To save code, let's use the ability of `input` to display a `prompt` (as shown in the docstring with `input?`:
  1. Put both programs in one code cell.
  2. Use `input` to ask for the `name` and the `age`.
  3. Print greeting with `name`, length of `name`.
  4. Print `age` next year.
  5. Sample run (terminal):
- Step 3: getting `input` with `prompt`:

```
print("Hello world!")
name = input("What is your name? ")
print("Good to meet you, " + name)
print("Your name has ", len(name), " characters")
age = input("What is your age? ")
print("You're going to be " + str(int(age) + 1) + " years old")
```

```
Hello, world!
What is your name? Marcus
Greetings, Marcus
Your name has 6 characters.
What is your age? 59
You are going to be 60 years old.
```

Figure 7: Testing input with prompt

14 Getting two input values at once

- Step 4: getting two input values at once with `split`:

```
print("Hello world!")
input_data = input("Enter name and age separated by a space: ")
name, age = input_data.split()
print("Good to meet you, " + name)
print("Your name has ", len(name), " characters")
print("You're going to be " + str(int(age) + 1) + " years old")
```

- Check out the docstring of this new function with: `split?`.
  - `split` is a string method - outside of `str` it has no meaning.
  - You have to look for `str.split?` to get the docstring.
  - Notice that `str.split()?` or `help(str.split())` throw errors.

15 Function preview

- Functions in your code are like mini programs. We called six functions: `print`, `input`, `len`, `int`, `str`, `split`:
  1. `print` prints its arguments but can also evaluate:

```
print("Hi")
print(5 + 5)
```

```
Hi
10
```

2. `input` takes input from the keyboard or from the command line (input stream `stdin`) and either prints it or lets you assign it to a variable (output stream `stdout`):

   ```
   input("What's your name? ") # prints and waits for input
   ```

3. `len` computes the length of its (string) argument and returns an integer:

   ```
   print(len("Birkenkrahe"))
   var = 'Dampfschiffahrtsgesellschaftskapitän'
   print(len(var))  # with the len() function
   print(var.__len__())  # with the str.__len__ method
   ```

   ```
   11
   37
   37
   ```

4. `str` returns its value as a string:

   ```
   print(str(1000) + " random numbers")
   print(str('1000') + " random numbers")
   ```

   ```
   1000 random numbers
   1000 random numbers
   ```

5. `split` returns a `list` of words that can be split up among different variables:

   ```
   name = "Marcus 2  Birkenkrahe"
   print(name.split()) # default: split on whitespace, ignore ' '
   first, last = name.split()  # split name in two parts
   print(first,last)
   print(first + last)
   ```

   ```
   ['Marcus', '2', 'Birkenkrahe']
   ```

16 A few open questions

- What does the expression `str(int(age) + 1)` do?

  1. `age` is string `input`
  2. `int(age)` converts the string to a number - you cannot do that with any character like "a": `int("a")` throws an error. To convert characters to their Unicode standard, you need to use `ord`:

     ```
     print(int("25"))
     print(ord("a"))
     print(ord("A"))
     ```

  3. `int(age) + 1` adds 1 to whatever number `int(age)` evaluates to:

     ```
     age = "25"
     print(age)
     print(age + " years old")
     print(int(age))
     print(int(age)+1)
     ```

     ```
     25
     25 years old
     25
     26
     ```

  4. `str(int(age) + 1)` converts the result to a string:

     ```
     age = "25"
     print(age)
     print(age + " years old")
     print(int(age))
     print(int(age)+1)
     print(str(int(age)+1))
     print(str(int(age)+1) + " years old")
     ```

     ```
     25
     25 years old
     25
     26
     26
     26 years old
     ```

- Here is an HTML animation to illustrate these steps (Sweigart, 2023)

- `split(self, / , sep=None, maxsplit=-1)` is a *string method* with two optional (defaulted) arguments - it returns list of words in the string using `sep` as the delimiter, at most `maxsplit` splits are done: elements (note the implicit arguments):

```
print('1,2,3'.split(','))  # default maxsplit = -1 means no limit
print('1,2,3'.split(',',0)) # don't split
print('1,2,3'.split(',',1)) # split once
print('1,2,3'.split(',',2)) # split twice
print('1,2,3'.split(',',3)) # split thrice - nothing more to do
```

- The dot-operator `.` is an *accessor*: it allows you to access anything that's stored inside an object, e.g. the *string* class `str`, or an instance of that class, a particular string.
- What happens when the string to be split does not have substrings?

```
a, b = 'Marcus'.split()
print(a,b)
```

- Why?

```
help(str.split)

Help on method_descriptor:

split(self, /, sep=None, maxsplit=-1)
    Return a list of the substrings in the string, using sep as the separator stri

      sep
        The separator used to split the string.

        When set to None (the default value), will split on any whitespace
        character (including \\n \\r \\t \\f and spaces) and will discard
        empty strings from the result.
      maxsplit
        Maximum number of splits (starting from the left).
        -1 (the default value) means no limit.

    Note, str.split() is mainly useful for data that has been intentionally
    delimited.  With natural text that includes punctuation, consider using
    the regular expression module.
```

- What does the `/` refer to in the `str.split` docstring:

  ```
  str.split(self, /, sep=None, maxsplit=-1)
  ```

  The `/` is a *parameter separator*: it denotes the end of positional-only parameters. After `self` (the string itself), the parameters `sep` and `maxsplit` can be explicitly assigned:

  ```
  print(str.split('Marcus Birkenkrahe'))
  print(str.split('Marcus_Birkenkrahe','_'))
  print(str.split('Marcus_Birkenkrahe',sep='_'))
  print('Marcus_Birkenkrahe'.split(sep='_'))
  print('Marcus_Birkenkrahe'.split('_'))

  ['Marcus', 'Birkenkrahe']
  ['Marcus', 'Birkenkrahe']
  ['Marcus', 'Birkenkrahe']
  ['Marcus', 'Birkenkrahe']
  ['Marcus', 'Birkenkrahe']
  ```

17 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
- Comments begin with a # character and are ignored by Python; they are notes & reminders for the programmer.
- Functions are like mini-programs in your program.

- The `print` function displays the value passed to it.
- The `input` function lets users type in a value.
- The `len` function takes a string value and returns an integer value of the string's length.
- The `int`, `str`, and `float` functions can be used to convert data.

18 Glossary

TERM/COMMAND	MEANING
expression	a basic programming instruction, like `2+2`
values	something stored in a computer memory cell
operator	a function that takes values to evaluate them
binary operator	an operator that takes 2 values as arguments
whitespace	empty space between values or operators
indentation	empty spaces at the beginning of a line
precedence	order of operations
Syntax error	you've broken the grammatical Python rules
Type error	you've made a mistake with data types
Concatenation	adding strings with +
Replication	replicating strings with *
Conversion	changing data types
Coercion	implicit conversion of data types
File type	used by the computer to identify a language
Data type	used by the computer to reserve memory
`print`	printing function
`input`	takes input from stdin (e.g. keyboard, file)
`len`	returns length of argument
`str.split`	splits string into substrings
`str.strip`	removes leading and trailing whitespace
`int`, `float`, `str`	data type conversion functions

19 References

- pythontutor.com (2023). Visualize code execution.
- 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.