CMSC330: Python Basics

Chris Kauffman

Last Updated:
Mon Aug 28 05:38:32 PM EDT 2023

Logistics

Reading

Goals

- Understand basic syntax of Python
- Relate Python to Java
- Identify imperative nature of both Languages

Python

- Development started in late 1980s
- Version 2 released in 2000, fairly recognizable
- Version 3 released in 2008, was NOT backwards compatible

```
print "Hello"   # version 2
print("Hello")  # version 3
```

- Created a vast schism; still some version 2 code in use out there today
- "Fun" to program in: do a lot with few lines of code
- Relatively straight-forward to interface with C
- Often used as an intro language due to its friendly looking syntax (both my old university did and UMD is rumored to be looking to try Python in 131)
- Wildly popular in all realms of computing from web frameworks to machine learning / data science to robotics, great to have on your resume

Python's Primary author is Dutch coder Guido von Rossum, dubbed "Benevolent dictator for life" by the development community.

Every Programming Language

Look for the following as it should almost always be there

- ☐ Comments
- ☐ Statements/Expressions
- ☐ Variable Types
- ☐ Assignment
- ☐ Basic Input/Output (printing and reading)
- ☐ Function Declarations
- ☐ Conditionals (if-else)
- ☐ Iteration (loops)
- ☐ Aggregate data (arrays, records, objects, etc)
- ☐ Library System

Exercise: Collatz Computation An Introductory Example

- `collatz.py` prompts for an integer and computes the Collatz Sequence starting there
- The current number is updated to the next in the sequence via

  ```
  if cur is EVEN cur=cur/2; else cur=cur*3+1
  ```
- This process is repeated until it converges to 1 (mysteriously) or the maximum iteration count is reached
- The code demonstrates a variety of Python features and makes for a great crash course intro
- With a neighbor, study this code and identify the features you should look for in every programming language

Exercise: Collatz Computation An Introductory Example

```python
# collatz.py: collatz computation
verbose = True                  # global var

def collatz(start,maxsteps):    # function
  cur = start
  step = 0
  if verbose:
    print("start:",start,"maxsteps:",maxsteps)
    print("Step  Current")
    print(f"{step:3}: {cur:5}")
  while cur != 1 and step < maxsteps:
    step += 1
    if cur % 2 == 0:
      cur = cur // 2
    else:
      cur = cur*3 + 1
    if verbose:
      print(f"{step:3}: {cur:5}")
  return (cur,step)

# executable code at global scope
start_str = input("Collatz start val:\n")
start = int(start_str)

(final,steps) = collatz(start, 500)
print(f"Reached {final} after {steps} iters")
```

Look for... Comments, Statements/Expressions, Variable Types, Assignment, Basic Input/Output, Function Declarations, Conditionals, Iteration, Aggregate Data, Library System

```
>> python collatz.py
Collatz start val:
10
start: 10 maxsteps: 500
Step  Current
  0:    10
  1:     5
  2:    16
  3:     8
  4:     4
  5:     2
  6:     1
Reached 1 after 6 iters
```

Answers: Collatz Computation An Introductory Example

- ☒ Comments: `# comment to end of line`
- ☒ Statements/Expressions: written plainly, no semicolons, stuff like `a+b` or `n+=2` is old hat; Boolean expressions available via `x and y` implicating `z or w` is likely around
- ☒ Variable Types: string, integer, boolean are obvious as values, no type names mentioned save the conversion from string to integer via the `int(str)` function
- ☒ Assignment: via `somevar = avalue`
- ☒ Basic Input/Output (printing and reading): `print() / input()`
- ☒ Function Declarations: `def funcname(param1,param2):`
- ☒ Conditionals (if-else): `if cond:` and `else:`, also `elif:`
- ☒ Iteration (loops): clearly `while cond:`, others soon
- ⊟ Aggregate data (arrays, records, objects, etc): `(python,has,tuples)` and others we'll discuss soon
- ☐ Library System: soon

A Few Oddities

- Python has two division operators `a / b` for floating point division, `a // b` for integer division. Dynamic types make this easy to forget and likely to cause errors

  ```
  >>> 11 / 3            # float div
  3.6666666666666665
  >>> 11 // 3           # int div
  3
  >> 11.99 / 3.99       # float div
  3.0050125313283207
  >>> 11.99 // 3.99     # what now?
  3.0
  ```

-

REPL: Read-Evaluate-Print Loop

- Python features a REPL to interactively interpret Python statements on the fly

The Whitespace Thing

Module / Namespace System

Python “main()” Functions

Built-in Data Types

Basic Control Flow Operations

For Loops and Iterators

Standard Scoping Rules

Nested Scopes / Nested Functions

Object Oriented Programming (OOP) Support

Dynamic Evaluation via `exec()`