Python Notes

Dylan Yu

August 8, 2020

Summary

These are my Python notes from a **very** long time ago. I have edited it to match my style file, and I think they will, at the very least, be an interesting read and serve as a nice "cheatsheet" for the easier parts of Python.

Contents

§ 1 Hardware

§ 1.1 Vocabulary

1. **Input and Output Devices:** The keyboard, the mouse, the screen; those are all input and output devices.

2. **Central Processing Unit (CPU):** Closest thing to intelligence. Has millions of transistors (set of electrical pulses through a wire).

3. **Main Memory:** When CPU says ”What’s next?”, main memory feeds its instructions into CPU.

4. **Secondary Memory:** Example: create a file. Feeds to main memory.

When the CPU keeps asking for instructions and the memory keeps feeding it, this is known as the **Fetch-Execute Cycle**.

§ 2 Basics

§ 2.1 Types of Code

§ 2.1.1 Sequential

Simply goes in order. Just normal statements - reads left to right, up to down.

§ 2.1.2 Repeated

For loops, while loops, etc.

§ 2.1.3 Conditional

If/else loops, where if the condition is met then something is done.

§ 2.2 Printing

This is pretty intuitive. The keyword ”print” allows you to print what you want in the parenthesis. Remember - quote marks are needed to denote a string literal, otherwise it will think *hello world* is a variable.

```
print("hello world")
```

§ 2.3 Input

```
input('What is your name?')
```

§ 2.4 Whitespace

Whitespace matters in Python, unlike Java and C++. Here you must indent using a tab. Four spaces is standard (even over tab). Never mix spaces and tabs. Be consistent on consecutive lines. Only deviate to **improve** readability.

§ 2.5 Importing

To import, simply use:

```
import module_name
```

In this case, `module_name` represents the name of the module. Try `import math` or `import this`!

```python
import math
math.sqrt(81)
```

This will return `9.0`. For a description of what the functions are and what they do, type `help(object)`, where `object` is the imported thing you need help with. In this case, the object is `math`:

```
import math
help(math)
```

This will return a list of functions and their definitions.

Pressing enter when the program states `More` will give you more functions. A few other key terms:

```
from module import name
from module import name as name2
```

The former allows you to import selected elements of the module, and the latter allows you to rename elements.

§ 2.6 Arithmetic

- The plus sign, `+`, represents addition or concatenation.
- The minus sign, `-`, represents subtraction.
- The asterisk, `*`, represents multiplication.
- The double asterisk, `**`, represents power. For example, `a**b` means a^b.
- The forward slash, `/`, represents real number division.
- The double forward slash, `//`, represents integer division.
- The percentage sign, `%`, represents modular arithmetic. For example, `a%b` means $a \pmod{b}$.

§ 2.7 Python REPL

The acronym REPL stands for read-evaluate-print-loop, and provides a programmer with an interactive programming environment. You activate the REPL by typing `python` or `$ python`, and exit using Ctrl-Z or Ctrl-D.

§ 2.8 Underscore

The underscore is assigned to the last value in the REPL.

§ 2.9 Scalar Types

- **int:** arbitrary precision integer
  - Are usually specified in decimal, but can be specified to be in binary, octal, and hexadecimal. For example, `0b10` will return `10` in binary, `0o10` will return `10` in octal, and `0x10` will return `16` in hexadecimal.
  - Putting in a non-integer real value into the `int()` function will round to the value closest to 0. For example, `int(3.5)` returns `3` and `int(-3.5)` returns `-3`.
  - `int("a",b)` will return a in base b
- **float:** IEEE-754 double-precision with 53-bits of binary precision (15-16 significant digits in decimals)
  - Note that `e` represents 10 to some power. For example, `3e10` means $3 \cdot 10^{10}$.
  - You can also typecast to a float. For example, `float("1.618")` casts to `1.618`, `float("nan")` type casts to `nan`, and `float("inf")` type casts to `inf` (`-inf` will do the same). Note that `nan` and `inf` are special - other strings cannot typecast.
- **NoneType:** the null object
  - Binds a value to `None`.
  - Can help determine if something is null
- **bool:** boolean logical values
  - There are two values for bool operators: `True` and `False`. Note that they are capitalized, unlike C++ and Java.
  - For integers, all integers besides 0 are considered truthy. Similarly, for floats, all decimals besides 0.0 are considerd truthy. Any form of 0 is falsy.
  - For lists, only the empty list is falsy (non-empty lists are truthy).
  - For strings, all statements that have something in them are truthy. Only `""` is falsy.

§ 2.10 Relational Operators

- `==` represents value equality / equivalence
- `!=` represents value inequality / inequivalence
- `<` represents less-than
- `>` represents greater-than
- `<=` represents less-than or equal
- `>=` represents greater-than or equal

§ 2.11 Control Flow

- If-statement: if true, it will enter the loop
- Else-statement: if it fails the condition for the if loop, it will go here (assuming no elif loops)
- Elif-statement: if it fails the condition for the if loop, it will go here, if this fails, it will go to the else loop
- Pass: `pass` doesn't do anything technically, but rather it is used when you want to implement later rather than right now.

§ 2.12 Loops

- While-loop: while the loop condition is satisfied, the loop will keep continuing. `break` puts the loop ending in a predicate test. This is seen in C++ and Java with do-while.