https://github.com/cheperuiz/ hackerdojo-pythonistas

Python 101: Hello, World! But first, lets address some common myths.

Q: Python is a scripting language, there

is no compiler!

Q: Python is a scripting language, there is no compiler!

R: Python is an interpreted language; each script is compiled (and cached) to bytecode during the first run. Execution happens inside a VM.

Q: Python is dynamically typed.

Q: Python is dynamically typed.

R: Objects in python are strongly typed, but names can be reassigned dynamically.

Python typing system is commonly called "Duck Typing" (if it quacks like a duck, walks like a duck and swims like a duck, you can assume it's a duck).

The type of the object is not as important as the methods it defines (__len__, __add__, __gt__, etc.).

Some things to know...

- In python everything is a pointer.
- One can only interact with an object through its methods.
- int, str & float are immutable types (no __set__() method).
- Integers can grow indefinitely (if there is memory available).
- All objects inherit from <object> and are created through factories (including functions & modules).
- Only built-in types have private members. All user class attributes are entirely public.

Q: Python is slow!

Q: Python is slow!

R: Yes.

Q: Python is slow!

R: But also, no:

- 1) Usually the 'slow code' can be profiled and re-implemented more efficiently (built-in tools like cprofile can help).
- 2) There are multiple interpreters & strategies to speed up your code (Intel® Distribution for Python, pypy, numba).
- 3) If execution speed is crucial, perhaps that module should be written in a performance-oriented language (C, C++, Rust). Or as an extension with a python API.

Q: Python code can't be multithreaded, and the GIL is a POS!

Q: Python code can't be multithreaded, and the GIL is a POS!

R: The Global Interpreter Lock (GIL) can be an obstacle in very specific (CPU-bound) cases, but there are several strategies to follow if you need concurrent or parallel tasks:

- 1) Use multiprocessing instead of multithreading.
- 2) Use async-await syntax.
- 3) Do you really need to use multithreading for this component?

 Perhaps you could use follow a different (safer) pattern like the Actor Model.

But first, MORE facts about python.

Quick notes about Python

- -Official stable version is Python 3.10
- Python 2.x is officially deprecated.
- There are some (major) breaking changes in 3.5-3.6, 3.6-3.7.
- The official interpreter is CPython but there are many compliant implementations.
- -Single source of truth: python.org

Initial goals of Guido Van Rossum (former BDFL*) for Python.

- An easy and intuitive language just as powerful as major competitors.
- Open source, so anyone can contribute to its development.
- Code that is as understandable as plain English.
- Suitability for everyday tasks, allowing for short development times.

*no longer BDFL because of the Walrus Operator RFC arguments.

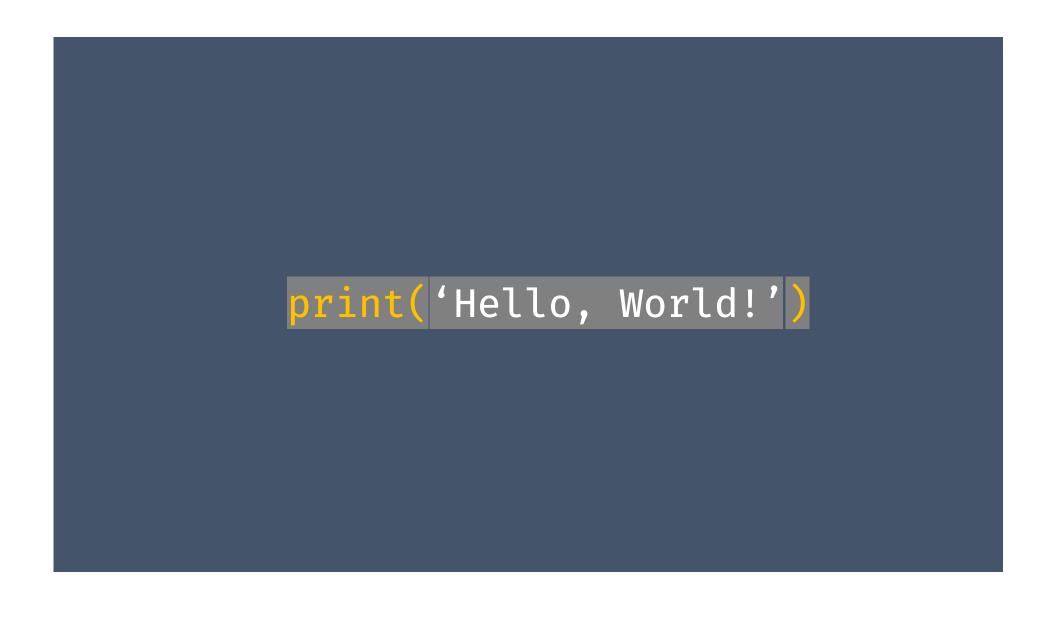
But first, let's make sure we write 'good Python'

Good Python? What's that?

- The Zen of Python, by Tim Peters.
- <u>PEP8 Style Guide</u> (snake_case for functions & variable names, CamelCase for classes).
- What you know from other languages, still apply:
 - Don't try to be clever...
 - Remember: Code is more often <u>READ</u> than <u>WRITTEN</u>.
 - Follow your principles:
 - DRY, SOLID
 - If you're having this problem in 2022, someone else has already implemented a solution...
 - Try to apply design patterns & idioms.

Functional, yet Pythonic

- Declarative.
 - Write 'what' task to perform, not 'how' to do it.
- Prefer 'pure functions':
 - No internal state.
 - Idempotent, with no side effects.
- Choose immutable types whenever possible.
- Use []/{}/() comprehensions over filter, reduce & map.
- Leverage the language built-in types & features.
- Prefer dependency injection.



print('Hello, World!')

Thanks :)