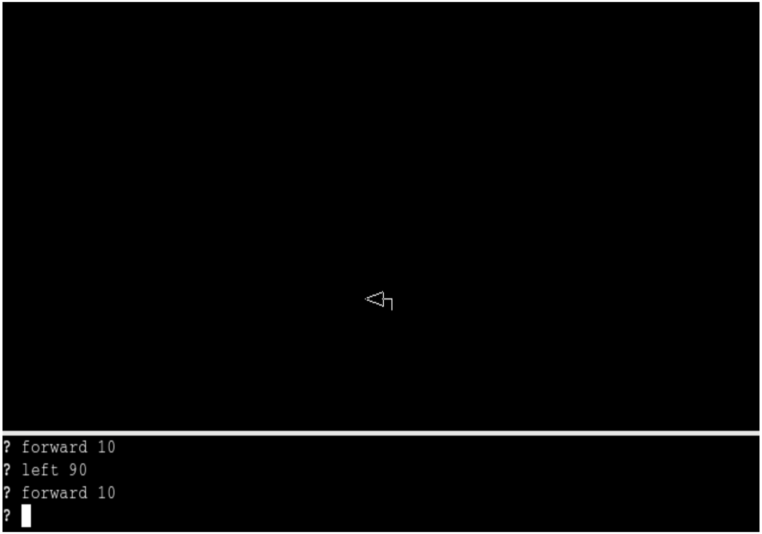
Exploration Oriented Programmming

REPL to Production

Moshe Zadka – https://cobordism.com

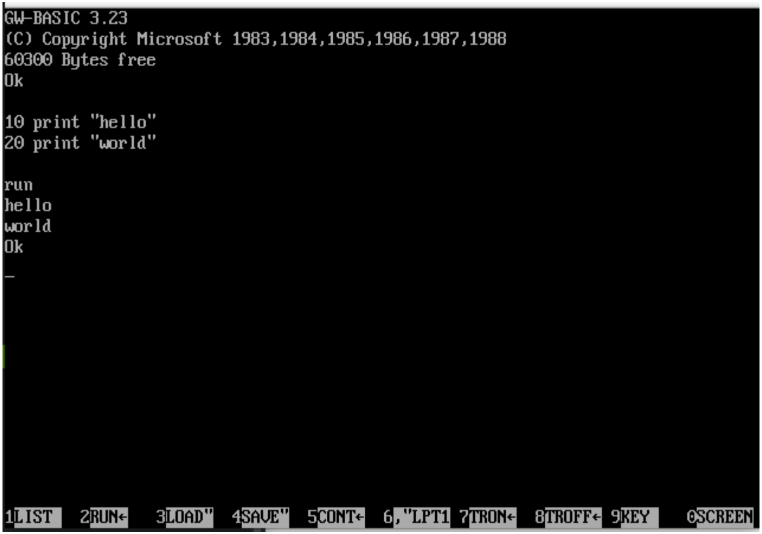
North Bay Python 2018

LOGO



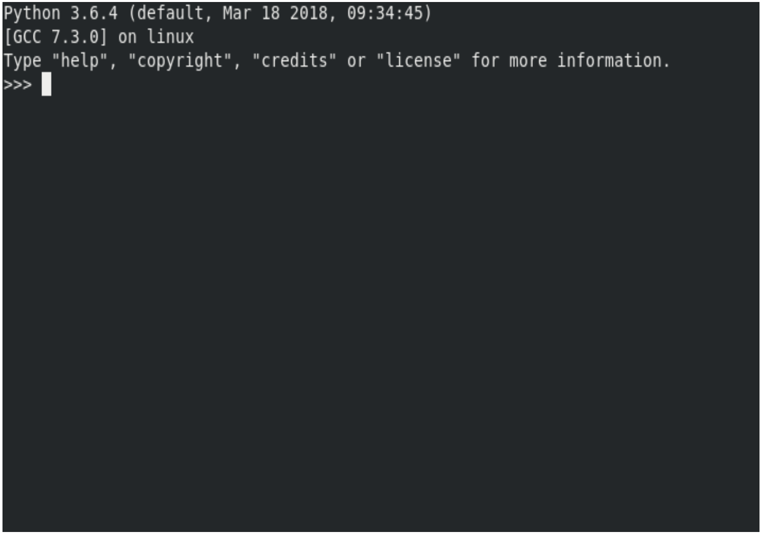
```
? forward 10
? left 90
? forward 10
? 
```

GW-Basic



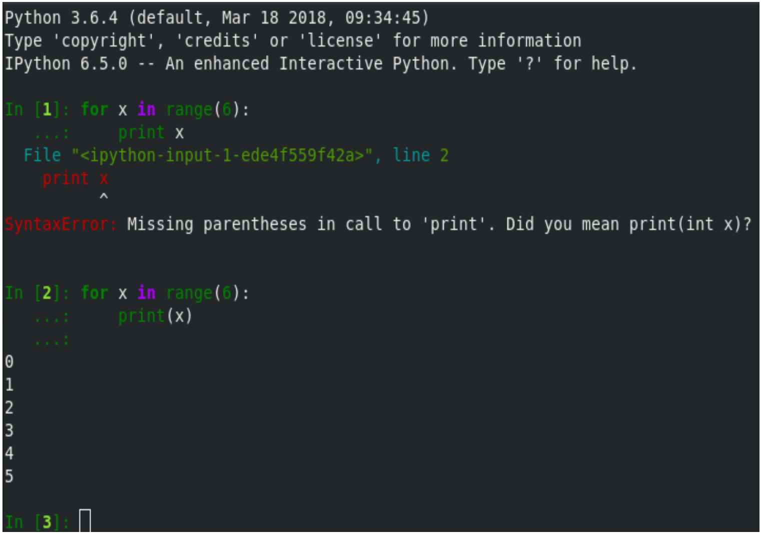
```
GW-BASIC 3.23
(C) Copyright Microsoft 1983,1984,1985,1986,1987,1988
60300 Bytes free
Ok

10 print "hello"
20 print "world"

run
hello
world
Ok

_
```

Python REPL



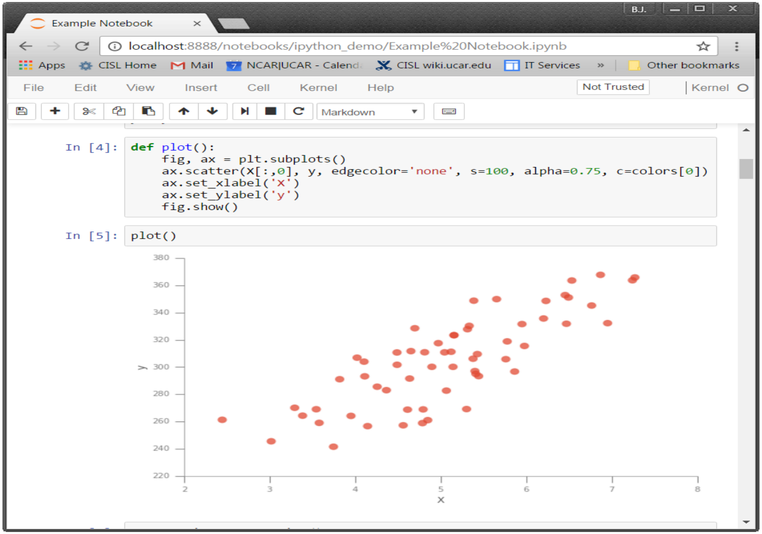
```
Python 3.6.4 (default, Mar 18 2018, 09:34:45)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

IPython REPL



```
Python 3.6.4 (default, Mar 18 2018, 09:34:45)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: for x in range(6):
   ...:     print x
  File "<ipython-input-1-ede4f559f42a>", line 2
    print x
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(int x)?


In [2]: for x in range(6):
   ...:     print(x)
   ...:
0
1
2
3
4
5

In [3]:
```

Jupyter



```
In [4]: def plot():
            fig, ax = plt.subplots()
            ax.scatter(X[:,0], y, edgecolor='none', s=100, alpha=0.75, c=colors[0])
            ax.set_xlabel('x')
            ax.set_ylabel('y')
            fig.show()

In [5]: plot()
```

What is Jupyter?

- Web interface
- Kernel
- Persistent history
- Other goodies!

Kernel

- Handles snippets
- In-memory state
- Semi-disposable
- Tornado event loop

Magic

```
%%pdb

%% capture output
...
```

Server vs. Kernel

One Jupyter server, many kernels.

Adding a Kernel

```
def add_to(kernel_venv, jupyter_venv):
    cc = subprocess.check_call
    p = os.path
    python = p.join(kernel_venv, 'bin', 'python')
    name = p.basename(kernel_venv)
    cc([python, '-m',
        'pip', 'install', 'ipykernel'])
    cc([python, '-m',
        'ipykernel', 'install',
        '--name', name,
        '--display-name', name,
        '--prefix', venv])
    spec = p.join(kernel_venv,
                  'share/jupyter/kernels', name)
    jupyter = p.join(jupyter_venv, 'bin/jupyter')
    cc([jupyter, 'kernelspec', 'install', spec])
```

Security Model

- Opaque security token
- By default, listen only on localhost

Notebooks

- Editable history
- Inputs and outputs
- Code, not state

Notebooks from the Inside

```
{
 "cells": [
   { "cell_type": "code",
     ...
      "source": ["1 + 1"]
   }
  ]
 "nbformat": 4,
 "nbformat_minor": 1
}
```

Global namespace

```
some_thing = 15
```

```
some_thing * 2
```

```
30
```

Redefining functions

```
def foo(a):
    return 2 * a
```

```
foo(10)
```

```
20
```

```
def foo(a):
    return 3 * a
```

```
30
```

Immutable data structures

```
a = v(1, 2, 3)
```

```
def increase_head(stuff):
    return stuff.set(0, stuff[0] + 1)
increase_head(a)
```

```
pvector([2, 2, 3])
```

```
def increase_tail(stuff):
    return stuff.set(-1, stuff[-1] + 1)
increase_tail(a)
```

```
pvector([1, 2, 4])
```

Verification as testing

```
# test
x = [1, 2, 3]
y = increase_tail(x)
assert_that(y[2], is_(5))
```

```
...
AssertionError:
Expected: <5>
     but: was <4>
```

Classes

```
@attr.s(frozen=True)
class Point:
    x = attr.ib()
    y = attr.ib()
```

Dispatching

```
@singledispatch
def abs(thing):
    raise NotImplementedError("No abs",
                              thing)
```

```
@abs.register(Point)
def abs(pt):
    return (pt.x**2 + pt.y**2) ** 0.5
```

Version control

```
"execution_count": 1,
"outputs": [
 {
  "data": {
   "text/plain": [
    "2"
   ]
  },
  "execution_count": 1,
  "metadata": {},
  "output_type": "execute_result"
 }
],
```

Cleaning outputs

```
with open("something.ipynb") as fpin:
    data = fpin.read()
    parsed = json.loads(data)
    for cell in parsed["cells"]:
        del cell["output"]
        del cell["execution_count"]
with open("something_cleaned.ipynb") as fpout:
    fpout.write(json.dumps(parsed))
```

Cleaning outputs

- Pre-commit hook
- Test in CI that re-cleaning gives same result
- Code review the cleaned file

Lint

```
% jupyter nbconvert --to=python something.ipynb
% flake8 something.py
```

Test

```
with open("something.ipynb") as fpin:
    notebook = json.loads(fpin.read())
with open("something.py", "w") as fpout:
    for cell in notebook["cells"]:
        if ("# pragma: interactive-only" in
                cell["source"]):
            continue
        fpout.write(f"\n{cell['source']}\n")
subprocess.check_output(["pytest", "something.py"])
```

Custom diff

```
# Suitable for use as "git difftool"
def to_lines(fname):
    with open(fname) as fpin:
        contents = json.loads(fpin.read())
    for i, cell in enumerate(contents["cells"]):
        yield f'Cell {i}'
        yield from cell["source"].splitlines()
sys.stdout.writelines(difflib.contextdiff(
    to_lines(os.environ['LOCAL']),
    to_lines(os.environ['REMOTE']),
    'a/' + os.environ['MERGED'],
    'b/' + os.environ['MERGED'],
))
```

Custom merge

- Clean
- Merge
- Add dummy output
- (Beyond current scope)

Importing Notebooks

```
@attr.s(frozen=True)
class NotebookLoader:
    contents = attr.ib()
    def create_module(self, spec):
        util = importlib.util
        return util.module_from_spec(spec)
    def exec_module(self, module):
        cells = json.loads(contents)["cells"]
        for cell in cells:
            if cell.starts_with("#pragma: module"):
                exec(cell, module.__dict__)
```

Finding Notebooks

```
class NotebookFinder(object):

    def find_module(self, fullname, path=None):
        if path is None:
            return None
        name = fullname.split('.')[-1] + '.ipynb'
        if not resources.is_resource(path, name):
            return None
        text = resources.read_text(path, name)
        return NotebookLoader(text)

import sys
sys.meta_path.append(NotebookFinder())
```

Integrating with packages

```
somepackage/
            __init__.py
              import sys
              sys.meta_path.append(NotebookFinder())
            module.ipynb
```

Producing documentation

```
.. automodule package.module
   :members:
```

Producing documentation

```
with open("something.ipynb") as fpin:
    notebook = json.loads(fpin)
with open("something.md", "w") as mdout:
    for cell in notebook["cells"]:
        if cell["cell_type"] != "markdown":
            continue
        mdout.write(cell["source"])
```

Building wheels

```
MANIFEST.in
  include *.ipynb
```

Exporting API

```
INTERACTIVE = False
```

```
# pragma: interactive-only
INTERACTIVE = True
```

```
from publication import publish
__all__ = ['some_function',
           'SomeClass']
if not INTERACTIVE:
    publish()
```

Code as Successive Approximation

Are we ever ”done”?

REPL as IDE

- Still nascent...
- ...getting better

Proud tradition

Lisp, Smalltalk, Logo, GW-Basic.