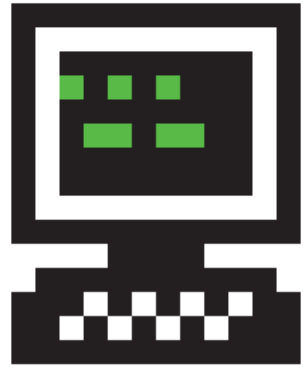
A tale of Python C extensions and cross-platform wheels  

EuroPython 2021

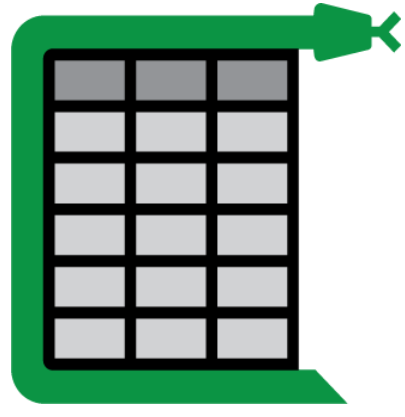
Vinayak Mehta

Outline

- A basic C extension module using the Python/C API
- The same C extension module using pybind11
- Wrapping a C++ library using pybind11
- Shared libraries and dynamic linking
- Building wheels and bundling shared libraries
- Automating wheel builds using cibuildwheel and Github Actions



https://www.recurse.com



https://camelot-py.readthedocs.io

PDF ➡ PNG

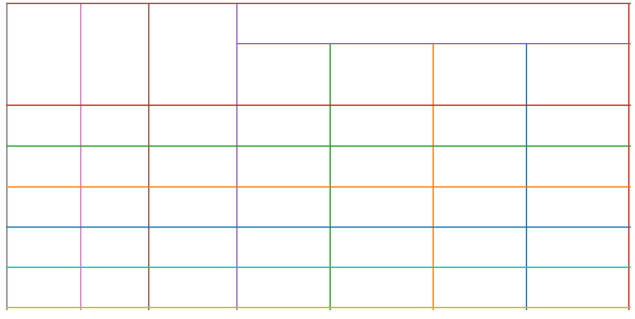


Table 2-1. Simulated fuel savings from isolated cycle improvements

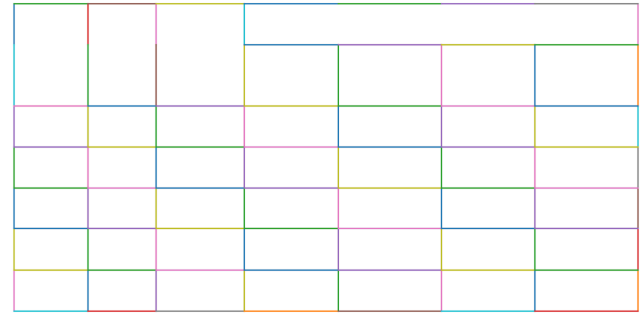
Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

2-1 extends the analysis from eliminating stops for the five example cycles and exam

Table 2-1. Simulated fuel savings from isolated cycle improvements

Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

2-1 extends the analysis from eliminating stops for the five example cycles and exam



The non-PyPI dependency problem

Seg fault 💥

ImportError 💥

"Is there a pure-Python PDF to PNG converter?"

poppler

https://github.com/freedesktop/poppler

pdftopng

```
>>> from pdftopng import pdftopng

>>> pdftopng.convert(pdf_path="foo.pdf", png_path="foo.png")
```

https://github.com/vinayak-mehta/pdftopng

Easier installation

```
🐧 sudo apt install ghostscript
🍎 brew install ghostscript
🪟 gs9540w64.exe
$ pip install camelot-py
```

➡

```
$ pip install camelot-py
```

on Windows, macOS, and Linux!

https://vinayak.io

The Python/C API

Use cases

- Writing extension modules
- Embedding Python in a another application

The example module

```
>>> from example import example
>>> example.add(1, 2)
3
>>> sum = example.add(1, 2)
>>> sum
3
```

src/example/example.c

```
#include "Python.h"
```

"Everything is ~~an object~~ a PyObject."

src/example/example.c

```
static PyObject*
add(PyObject *self, PyObject *args)
{

}
```

src/example/example.c

```c
static PyObject*
add(PyObject *self, PyObject *args)
{
  long a, b;

  if (!PyArg_ParseTuple(args, "ll:add", &a, &b)) {
    return NULL;
  }
}
```

src/example/example.c

```c
static PyObject*
add(PyObject *self, PyObject *args)
{
  long a, b;

  if (!PyArg_ParseTuple(args, "ll:add", &a, &b)) {
    return NULL;
  }

  PyObject *pA, *pB;
  pA = PyLong_FromLong(a);
  pB = PyLong_FromLong(b);

  PyObject *r = PyNumber_Add(pA, pB);
```

src/example/example.c

```c
static PyObject*
add(PyObject *self, PyObject *args)
{
  long a, b;

  if (!PyArg_ParseTuple(args, "ll:add", &a, &b)) {
    return NULL;
  }

  PyObject *pA, *pB;
  pA = PyLong_FromLong(a);
  pB = PyLong_FromLong(b);

  PyObject *r = PyNumber_Add(pA, pB);
```

src/example/example.c

```
static PyMethodDef
module_functions[] = {
  { "add", add, METH_VARARGS, "Add two numbers." },
  { NULL }
};
```

src/example/example.c

```c
static struct PyModuleDef example =
{
  PyModuleDef_HEAD_INIT,
  "example",
  "A minimal module.",
  -1,
  module_functions
};
```

src/example/example.c

```
PyMODINIT_FUNC PyInit_example(void)
{
  return PyModule_Create(&example);
}
```

setup.py

```
import os
from setuptools import setup, Extension

ext_modules = [
    Extension(
        "example.example",
        sources=[os.path.join("src", "example", "example.c")]
    )
]

setup(ext_modules=ext_modules)
```

```
$ pip install .
Successfully installed example-0.1.0
```

```
>>> from example import example
>>> example
<module 'example.example' from 'site-packages/example/exampl
```

```
>>> from example import example
>>> example.add(1, 2)
3
```

```
>>> from example import example
>>> example.add(1, 2)
3
>>> sum = example.add(1, 2)
>>> sum
3
```

pybind11 — Seamless operability between C++11 and Python

src/example/example.cpp

```
#include <pybind11/pybind11.h>
```

src/example/example.cpp

```
long add(long a, long b) {
  return a + b;
}
```

src/example/example.cpp

```
PYBIND11_MODULE(example, m) {
  m.doc() = "A minimal module.";
  m.def("add", &add, "Add two numbers.", py::arg("a"), py::a
}
```

setup.py

```
import os
import pybind11
from setuptools import setup, Extension

ext_modules = [
    Extension(
        "example.example",
        sources=[os.path.join("src", "example", "example.cpp
        include_dirs=[
            pybind11.get_include(),
        ],
        language="c++",
    ),
]
```

setup.py

```
setup(ext_modules=ext_modules)
```

pyproject.toml

```
[build-system]
requires = ["setuptools>=40.6.0", "pybind11>=2.5.0", "wheel"
build-backend = "setuptools.build_meta"
```

```
$ pip install .
Successfully installed example-0.1.0
```

```
>>> from example import example
>>> example
<module 'example.example' from 'site-packages/example/exampl
```

```
>>> from example import example
>>> example.add(1, 2)
3
>>> sum = example.add(1, 2)
>>> sum
3
```

Wrap all the things! 

- Easy to wrap a C/C++ library
- Less verbose compared to playing with PyObjects

Wrapping pdftoppm from poppler

pdftoppm.cc

```
// poppler headers
// pdftoppm constants
// pdftoppm variables
// pdftoppm functions
int main(int argc, char *argv[])
{
  // convert pdf to png
  return 0
}
```

pdftopng.cpp

```
// poppler headers

#include <pybind11/pybind11.h>
namespace py = pybind11;

// pdftoppm constants
// pdftoppm variables
// pdftoppm functions
void convert(char *pdfFilePath, char *pngFilePath)
{
  // convert pdf to png
}

PYBIND11_MODULE(pdftopng, m) {
```

```
>>> from pdftopng import pdftopng

>>> pdftopng.convert(pdf_path="foo.pdf", png_path="foo.png")
```

Shared libraries and dynamic linking

```c
int main() {
    printf("%s\n", "Hello world!");
}
```

```
$ gcc program.c -o program
$ ./program
```

```
$ ./program

  printf("Hello world!")

  "I need printf from the C standard library"
```

```
$ ./program
  "I need printf from libc."          ->   ld.so
  "Can you tell me where libc is?"
```

```
$ ./program

  "..."                    —>   ld.so
                                "Sure, let me look for it."
  "Thanks, I'll wait."
```

```
$ ./program
  "..."                    ->   ld.so   ->   libc.so
  "Thanks, I'll wait."          "Yay found it!"
```

```
$ ./program
  "..."                  <-   ld.so   <-   libc.so
  "..."                       "Here you go."
  "Thanks! I can now finish executing!"
  Hello world!
$
```

`libc.so` = Shared library

The *process* = Dynamic linking

Linux `.so` Search Order

- The default directories, normally `/lib` and `/usr/lib`
- The directories listed in `/etc/ld.so.conf`
- The environment variable `LD_LIBRARY_PATH`
- ...

Run `man ld` for full list.

Windows `.dll` Search Order

- `if dll_name in memory or dll_name in known_dlls: return False`
- Otherwise search for the DLL in this order:
  - ⬇ The directory from which the application was loaded
  - ⬇ The system directory
  - ⬇ The Windows directory
  - ⬇ The current directory
  - ⬇ The directories listed in the PATH environment variable

Standard Search Order for Desktop Applications

libfunc.so

mod1.c

```c
#include "stdio.h"

void mod1_func() {
    printf("mod1 says hello!\n");
}
```

```
$ gcc mod1.c mod2.c mod3.c -shared -o libfunc.so
```

prog.c

```c
#include "stdio.h"

void mod1_func();
void mod2_func();
void mod3_func();

void main() {
    mod1_func();
    mod2_func();
    mod3_func();
    printf("Hello world!");
}
```

```
$ gcc prog.c libfunc.so -o prog
```

```
$ ./prog
libfunc.so: No such file or directory
```

```
$ LD_LIBRARY_PATH=. ./prog
mod1 says hello!
mod2 says hello!
mod3 says hello!
Hello world!
```

Building wheels 

Part 1: Building shared libraries

Building shared libraries

- Linux
- macOS
- Windows

Linux

~~Linux~~ Manylinux

Manylinux shared library subset

```
libc.so.6
libdl.so.2
libgcc_s.so.1
libGL.so.1
libglib-2.0.so.0
libgobject-2.0.so.0
libgthread-2.0.so.0
libICE.so.6
libm.so.6
libnsl.so.1
libpthread.so.0
libresolv.so.2
librt.so.1
libSM.so.6
```

https://quay.io/organization/pypa

```
$ docker pull quay.io/pypa/manylinux<version>
$ docker run --rm -it -v $(pwd):/usr/src/project quay.io/pyp
```

```
$ yum install -y wget freetype-devel fontconfig-devel libpng
$ cmake .. && make poppler
Successfully built poppler
$ ls
libpoppler.so
```

build_linux.sh

```
yum install -y wget freetype-devel fontconfig-devel libpng-d
cmake .. && make poppler
```

macOS

fastmac

https://github.com/fastai/fastmac

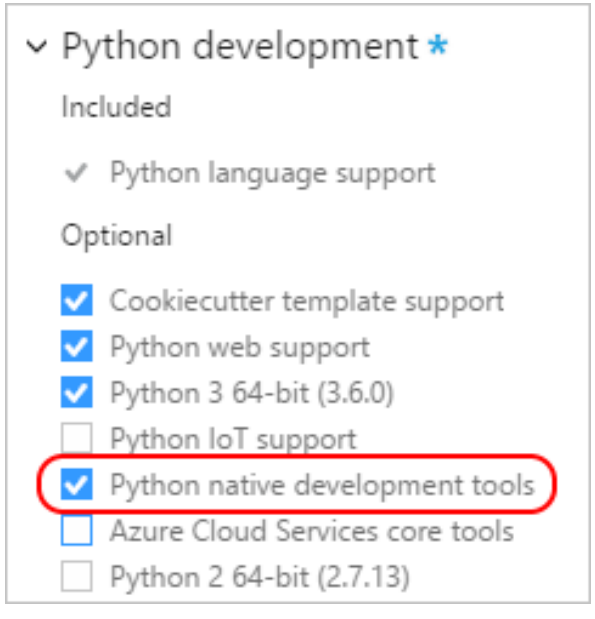
```
$ brew install pkg-config freetype fontconfig libpng jpeg
$ cmake .. && make poppler
Successfully built poppler
$ ls
libpoppler.so
```

build_macos.sh

```
brew install pkg-config freetype fontconfig libpng jpeg
cmake .. && make poppler
```

Windows

Visual Studio 2019 (or later) Community Edition



vcpkg — C++ Library Manager for Windows, Linux, and MacOS

https://github.com/microsoft/vcpkg

```
$ vcpkg install freetype:x64-windows fontconfig:x64-windows
$ cmake -A x64 -DCMAKE_TOOLCHAIN_FILE=%VCPKG_INSTALLATION_RO
$ make poppler
Successfully built poppler
$ ls
poppler.dll
```

build_win_x64.bat

```
vcpkg install freetype:x64-windows fontconfig:x64-windows li
cmake -A x64 -DCMAKE_TOOLCHAIN_FILE=%VCPKG_INSTALLATION_ROOT
make poppler
```

Linking our extension with the shared library

setup.py

```
if sys.platform in ["linux", "darwin"]:
    library_dirs = [
        os.path.join(os.getcwd(), "lib", "poppler", "build")
    ]
    libraries = ["poppler"]
```

setup.py

```
if sys.platform == "win32":
    library_dirs = [
        os.path.join(
            os.environ["VCPKG_INSTALLATION_ROOT"],
            "installed",
            "x64-windows",
            "lib"
        ),
        os.path.join(os.getcwd(), "lib", "poppler", "build")
    ]
    libraries = [
        "freetype",
        "fontconfig",
        "libpng16",
```

setup.py

```
include_dirs = [
    poppler_dir,
    os.path.join(poppler_dir, "fofi"),
    os.path.join(poppler_dir, "goo"),
    os.path.join(poppler_dir, "utils"),
    os.path.join(poppler_dir, "poppler"),
    build_dir,
    os.path.join(build_dir, "utils"),
    os.path.join(build_dir, "poppler"),
    pybind11.get_include(),
]
```

setup.py

```
ext_modules = [
    Extension(
        "pdftopng.pdftopng",
        sources=[os.path.join("src", "pdftopng", "pdftopng.c
        include_dirs=include_dirs,
        library_dirs=library_dirs,
        libraries=libraries,
        language="c++",
    ),
]
```

setup.py

```
setup(ext_modules=ext_modules)
```

```
$ pip install .
Successfully installed pdftopng-0.1.0
```

```
>>> from pdftopng import pdftopng

>>> pdftopng.convert(pdf_path="foo.pdf", png_path="foo.png")
```

```
$ pip wheel .
Created wheel for pdftopng: filename=pdftopng-0.2.3-cp38-cp3
Successfully built pdftopng
```

```
$ unzip -l pdftopng-0.2.3-cp38-cp38-manylinux2010_x86_64.whl
pdftopng/__init__.py
pdftopng/pdftopng.cpp
pdftopng/pdftopng.cpython-38-x86_64-linux-gnu.so
```

Part 2: Bundling shared libraries

Linux

auditwheel

https://github.com/pypa/auditwheel

```
$ auditwheel repair -w wheelhouse/ pdftopng-0.2.3-cp38-cp38-
```

```
$ LD_LIBRARY_PATH=$(pwd)/lib/poppler/build:$LD_LIBRARY_PATH
  auditwheel repair -w wheelhouse/ pdftopng-0.2.3-cp38-cp38-
```

```
$ unzip -l pdftopng-0.2.3-cp38-cp38-manylinux2010_x86_64.whl
pdftopng/__init__.py
pdftopng/pdftopng.cpp
pdftopng/pdftopng.cpython-38-x86_64-linux-gnu.so
pdftopng.libs/libz-eb09ad1d.so.1.2.3
pdftopng.libs/libfreetype-20bfc0cd.so.6.3.22
pdftopng.libs/libexpat-64fa60ba.so.1.5.2
pdftopng.libs/libjpeg-7feae879.so.62.0.0
pdftopng.libs/libpoppler-dba2df61.so.111.0.0
pdftopng.libs/libpng12-640ca796.so.0.49.0
pdftopng.libs/libfontconfig-63352676.so.1.4.4
```

macOS

delocate

https://github.com/matthew-brett/delocate

```
$ delocate-listdeps pdftopng-0.2.3-cp38-cp38-macosx_10_9_x86
  delocate-wheel -w wheelhouse/ -v pdftopng-0.2.3-cp38-cp38-
```

```
$ DYLD_LIBRARY_PATH=$(pwd)/lib/poppler/build:$DYLD_LIBRARY_P
  delocate-listdeps pdftopng-0.2.3-cp38-cp38-macosx_10_9_x86
  delocate-wheel -w wheelhouse/ -v pdftopng-0.2.3-cp38-cp38-
```

```
$ unzip -l pdftopng-0.2.3-cp38-cp38-macosx_10_9_x86_64.whl
pdftopng/__init__.py
pdftopng/pdftopng.cpp
pdftopng/pdftopng.cpython-38-darwin.so
pdftopng/.dylibs/libpng16.16.dylib
pdftopng/.dylibs/libfreetype.6.dylib
pdftopng/.dylibs/libjpeg.9.dylib
pdftopng/.dylibs/libfontconfig.1.dylib
pdftopng/.dylibs/libtiff.5.dylib
pdftopng/.dylibs/libpoppler.111.0.0.dylib
```

Windows

Windows `.dll` Search Order

- `if dll_name in memory or dll_name in known_dlls: return False`
- Otherwise search for the DLL in this order:
  - ⬇ The directory from which the application was loaded
  - ⬇ The system directory
  - ⬇ The Windows directory
  - ⬇ The current directory
  - ⬇ The directories listed in the PATH environment variable

Standard Search Order for Desktop Applications

The `package_data` way

setup.py

```
def copy_dlls():
    vcpkg_bin_dir = os.path.join(
        os.environ["VCPKG_INSTALLATION_ROOT"],
        "installed",
        "x64-windows",
        "bin"
    )
    for file in glob.glob(os.path.join(vcpkg_bin_dir, "*.dll
        shutil.copy(file, os.path.join("src", "pdftopng"))
```

setup.py

```
package_data = {}
if sys.platform == 'win32':
    copy_dlls()
    package_data = {'pdftopng': ['*.dll']}

setup(
    ext_modules=ext_modules,
    package_data=package_data
)
```

```
$ unzip -l pdftopng-0.1.0-cp38-cp38-win_amd64.whl
pdftopng/__init__.py
pdftopng/pdftopng.cpp
pdftopng/pdftopng.cp38-win_amd64.pyd
pdftopng/brotlicommon.dll
pdftopng/brotlidec.dll
pdftopng/brotlienc.dll
pdftopng/bz2.dll
pdftopng/freetype.dll
pdftopng/jpeg62.dll
pdftopng/libpng16.dll
pdftopng/zlib1.dll
```

The `__init__.py` shim

- `windll.LoadLibrary(dll_filepath)`
- `os.add_dll_directory(dll_directory)` (since Python 3.8)

DLL Hell 

The DLL mangling way

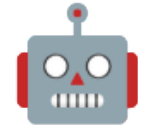
- Unpack the wheel
- Look for the extension module DLL
- Recursively look for DLL dependencies for the extension module
- Mangle the DLL names using their sha256 hash and copy them into the same directory as the extension module
- Modify the import table for extension module and each DLL with the new mangled DLL names using `machomachomangler`
- Zip the distribution directory with mangled DLLs into a wheel!

https://github.com/njsmith/machomachomangler

```
$ unzip -l pdftopng-0.1.0-cp38-cp38-win_amd64.whl
pdftopng/__init__.py
pdftopng/pdftopng.cpp
pdftopng/pdftopng.cp38-win_amd64.pyd
pdftopng/brotlicommon-7a839e03.dll
pdftopng/brotlidec-9899ad4b.dll
pdftopng/bz2-ee20a61d.dll
pdftopng/freetype-5999cc80.dll
pdftopng/jpeg62-357ce973.dll
pdftopng/libpng16-4b5cd968.dll
pdftopng/zlib1-1d35b9b6.dll
```

delvewheel

https://github.com/adang1345/delvewheel

Automating wheel builds using cibuildwheel and Github Actions 

cibuildwheel

https://github.com/pypa/cibuildwheel

.github/workflows/build_and_upload.yml

```yaml
name: build and upload

on:
  push:
    tags:
      - "*"
```

.github/workflows/build_and_upload.yml

```
env:
  CIBW_BUILD: "cp3?-manylinux_x86_64 cp3?-macosx_x86_64 "cp3
  CIBW_SKIP: "cp35-*"
```

.github/workflows/build_and_upload.yml

```
env:
  ...
  CIBW_BEFORE_BUILD_LINUX: "sh scripts/build_linux.sh"
  CIBW_REPAIR_WHEEL_COMMAND_LINUX: "LD_LIBRARY_PATH=$(pwd)/l
```

.github/workflows/build_and_upload.yml

```
env:
  ...
  CIBW_BEFORE_BUILD_MACOS: "sh scripts/build_macos.sh"
  CIBW_REPAIR_WHEEL_COMMAND_MACOS: "DYLD_LIBRARY_PATH=$(pwd)
```

.github/workflows/build_and_upload.yml

```
env:
  ...
  CIBW_BEFORE_BUILD_WINDOWS: call scripts\build_win_x64.bat
  CIBW_REPAIR_WHEEL_COMMAND_WINDOWS: "pip install delvewheel
```

.github/workflows/build_and_upload.yml

```yaml
jobs:
  build_wheels:
    name: Build wheels on ${{ matrix.os }}
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-18.04, macos-latest, windows-latest]
```

.github/workflows/build_and_upload.yml

```yaml
steps:
- uses: actions/checkout@v2
  with:
    submodules: true
```

.github/workflows/build_and_upload.yml

```
steps:
...
- uses: actions/setup-python@v2
  name: Install Python
  with:
    python-version: 3.8
```

.github/workflows/build_and_upload.yml

```
steps:
...
- uses: ilammy/msvc-dev-cmd@v1
  with:
    arch: amd64
- name: Build dependencies & wheels (Windows / amd64)
  if: runner.os == 'Windows'
  shell: cmd
  run: |
    python -m pip --disable-pip-version-check install cibuil
    python -m cibuildwheel --output-dir wheelhouse
```

.github/workflows/build_and_upload.yml

```
steps:
...
- name: Install cibuildwheel & build wheels (Linux & MacOS)
  if: runner.os != 'Windows'
  run: |
    python -m pip --disable-pip-version-check install cibuil
    python -m cibuildwheel --output-dir wheelhouse
```

.github/workflows/build_and_upload.yml

```
steps:
...
- uses: actions/upload-artifact@v2
  with:
    path: ./wheelhouse/*.whl
```

.github/workflows/build_and_upload.yml

```
upload_pypi:
  needs: [build_wheels]
  runs-on: ubuntu-latest
  # upload to PyPI on every tag starting with 'v'
  if: github.event_name == 'push' && startsWith(github.event
```

.github/workflows/build_and_upload.yml

```yaml
steps:
- uses: actions/download-artifact@v2
  with:
    name: artifact
    path: dist
```

.github/workflows/build_and_upload.yml

```
steps:
...
- uses: pypa/gh-action-pypi-publish@master
  with:
    user: __token__
    password: ${{ secrets.pypi_password }}
```

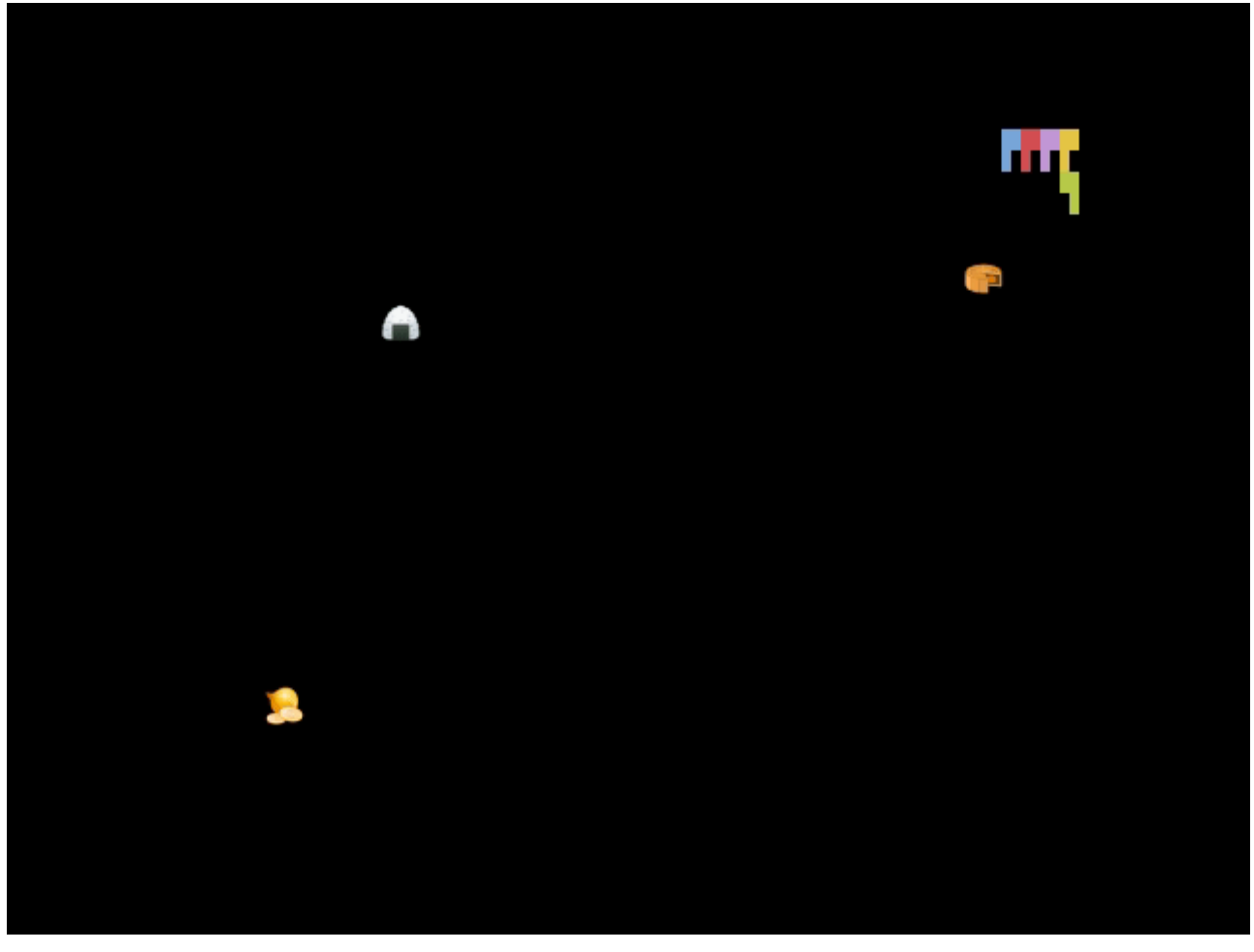
pdftopng

pdftopng-0.2.3-cp38-cp38-macosx_10_9_x86_64.whl (2.1 MB)	Wheel	cp38	Jul 5, 2021	View
pdftopng-0.2.3-cp38-cp38-manylinux2010_x86_64.whl (11.7 MB)	Wheel	cp38	Jul 5, 2021	View
pdftopng-0.2.3-cp38-cp38-win32.whl (1.3 MB)	Wheel	cp38	Jul 5, 2021	View
pdftopng-0.2.3-cp38-cp38-win_amd64.whl (1.5 MB)	Wheel	cp38	Jul 5, 2021	View
pdftopng-0.2.3-cp39-cp39-macosx_10_9_x86_64.whl (2.1 MB)	Wheel	cp39	Jul 5, 2021	View
pdftopng-0.2.3-cp39-cp39-manylinux2010_x86_64.whl (11.7 MB)	Wheel	cp39	Jul 5, 2021	View
pdftopng-0.2.3-cp39-cp39-win32.whl (1.3 MB)	Wheel	cp39	Jul 5, 2021	View
pdftopng-0.2.3-cp39-cp39-win_amd64.whl (1.5 MB)	Wheel	cp39	Jul 5, 2021	View

https://pypi.org/project/pdftopng

Fin.

curlyboi



https://github.com/vinayak-mehta/curlyboi

Code

- https://github.com/vinayak-mehta/python-ext-example
- https://github.com/vinayak-mehta/pybind11-ext-example
- https://github.com/vinayak-mehta/pdftopng
- https://github.com/vinayak-mehta/curlyboi
- https://github.com/numpy/numpy/wiki/windows-dll-notes

Talks

- Memory Management in Python - The Basics - Nina Zakharenko - PyCon US 2016
- A Whirlwind Excursion through Python C Extensions - Ned Batchelder - PyCon US 2009
- Here be Dragons - Writing Safe C Extensions - Paul Ross - PyCon US 2016
- Reliably Distributing Compiled Modules - Paul Kehrer - PyCon US 2016
- The Black Magic of Python Wheels - Elana Hashman - PyCon US 2019

Thank you!

@vortex_ape

vinayak.io