Generating a Python3 module to implement a fraction data type using gm2

- Prerequisites

- ```
  $ sudo apt install swig python3-dev
  ```

- you also need `gm2` version >= 15
  ```

Generating a Python3 module to implement a fraction data type using gm2

```
$ git clone https://github.com/gaiusm/pyfraction
$ cd pyfraction
$ mkdir build
$ cd build
$ ../configure
$ make
$ ./localrun.sh ../testfract.py
```

- `./localrun.sh` is a simple helper shell script to set `PYTHONPATH` and pass all arguments to the `python3` interpreter

The output from the ./localrun.sh session

```
PI =  3 1/7
one = 1
two = 2
three_and_half = 3 1/2
copy = 3 1/2
result = 1
1 + 2 = 3
3 1/2 = 3 1/2
1 + 3 1/2 = 4 1/2
1/4 + 3 1/2 = 3 3/4
1/4 + 1/3 = 7/12
1/7 + 1/3 = 10/21
60
10
60 * 10 = 600
6/7 * 3/8 = 9/28
```

testfract.py

```
#!/usr/bin/env python3

from fraction import *


def main ():
    p = PI ()
    print ("PI = ", p)
    one = fraction ().assign (1) ;
    two = fraction ().assign (2) ;
    three_and_half = fraction ().assign (3, 1, 2)
    print ("one =", one)
    print ("two =", two)
    print ("three_and_half =", three_and_half)
```

testfract.py

```
    copy = three_and_half
    print ("copy =", copy)
    result = one
    print ("result =", result)
    result += two
    print ("1 + 2 =", result)
    print ("3 1/2 =", three_and_half)
    print ("1 + 3 1/2 =", one + three_and_half)
    quarter = fraction ().assign (0, 1, 4)
    print ("1/4 + 3 1/2 =", quarter + three_and_half)
    third = fraction ().assign (0, 1, 3)
    print (quarter, "+", third, "=", quarter + third)
```

testfract.py

```
third = fraction ().assign (0, 1, 3)
seventh = fraction ().assign (0, 1, 7)
print (seventh, "+", third, "=", seventh + third)
val = fraction ().assign (60)
# fpc = fraction ().assign (0, 4, 10)
fpc = fraction ().assign (10)
print (val)
print (fpc)
print (val, "*", fpc, "=", val * fpc)
sixsevenths = fraction ().assign (0, 6, 7)
threeeights = fraction ().assign (0, 3, 8)
print (sixsevenths, "*", threeeights, "=", sixsevenths * threeeights)
```

Makefile.am rules generating the shared library _fractapi.so

```
SUFFIXES = .c .mod .def .o .obj .lo .a .i
DEPFLAGS=-MMD -MT $@ -MP -MF .deps/$*.d

SHELL=/bin/bash

prefix= @prefix@
datarootdir= @datarootdir@
srcdir= @srcdir@
mandir= @mandir@
exec_prefix=@exec_prefix@
bindir= @bindir@
INSTALL_PROGRAM = install

PYTHON_VERSION=$(strip $(shell python3 -V 2>&1 | \
  cut -b8- | cut -f1-2 -d'.'))

noinst_LTLIBRARIES = libfractapi.la

libfractapi_la_SOURCES = Fractions.mod fractapi.mod GC.mod

M2_DEPS = Fractions.lo fractapi.lo GC.lo
```

Makefile.am rules generating the shared library _fractapi.so

```
GM2_LIB_DIR=$(shell gm2 -print-file-name=)
# GCC_DEFAULT_LIB=/opt/gm2/lib      # for arm
GCC_DEFAULT_LIB=$(HOME)/opt/lib64

SRC_PATH_PIM=-I$(srcdir)/pge-m2  \
  -fm2-pathname=m2pim -I$(GM2_LIB_DIR)/m2/m2pim  \
  -fm2-pathname=m2iso -I$(GM2_LIB_DIR)/m2/m2iso  \
  -fno-m2-pathname=-

PROFILE =
CHECK = -fsoft-check-all
# CHECK =
OPT = -O0
```

Makefile.am rules generating the shared library _fractapi.so

```
GM2FLAGS = $(OPT) $(PROFILE) $(CHECK) -g -fm2-g  \
 -fiso -fextended-opaque \
 -I$(srcdir)/src-m2 -g
```

Makefile.am rules generating the shared library _fractapi.so

```
_fractapi.so: fractapi.lo
        cp fractapi.lo _fractapi.so
```

Makefile.am rules generating the shared library _fractapi.so

```
fractapi.lo: fractapi.mod
        $(LIBTOOL) --tag=CC $(AM_LIBTOOLFLAGS) \
          $(LIBTOOLFLAGS) --mode=compile gm2 -c \
          $(CFLAGS_FOR_TARGET) $(LIBCFLAGS) \
          $(libgm2_la_M2FLAGS) $(CHECK) \
          -g -fm2-g -fno-m2-pathname=- \
          -I$(srcdir) $(SRC_PATH_PIM) \
          -fruntime-modules=- \
          -fgen-module-list=fractapi.lst \
          -fscaffold-main \
          -fscaffold-static -fshared \
          $(srcdir)/fractapi.mod -o $@
```

Makefile.am rules generating the shared library _fractapi.so

```
libfractapi.la: $(M2_DEPS)
        gm2 -fswig -c -I$(srcdir) $(srcdir)/fractapi.mod
        swig -outdir . -o fractapi_wrap.cxx -c++ -python fractapi.i
        $(LIBTOOL) --tag=CC --mode=compile g++ -g -c \
          fractapi_wrap.cxx \
          -I/usr/include/python$(PYTHON_VERSION) \
          $(CFLAGS_FOR_TARGET) -o fractapi_wrap.lo
        $(LIBTOOL) --tag=CC --mode=link gcc -g $(M2_DEPS) \
          fractapi_wrap.lo -L$(GCC_DEFAULT_LIB) \
          -rpath `pwd` -lm2iso -lm2pim -lgcc -lstdc++ \
          -lc -lm -o libfractapi.la
        cp .libs/libfractapi.so _fractapi.so
```

Makefile.am rules generating the shared library _fractapi.so

```
%.lo: %.mod .deps/%.d
        @test -z .deps || mkdir -p .deps
        $(LIBTOOL) --tag=CC $(AM_LIBTOOLFLAGS) $(LIBTOOLFLAGS) \
          --mode=compile \
        gm2 -g -fm2-g -fiso $(DEPFLAGS) $(OPT) $(CHECK) \
          -fextended-opaque -I. -I$(srcdir) -c $< -o $@

.c.lo:
        $(LIBTOOL) --tag=CC $(AM_LIBTOOLFLAGS) $(LIBTOOLFLAGS) \
          --mode=compile \
        gcc -c $(DEPFLAGS) $(CFLAGS_FOR_TARGET) $(LIBCFLAGS) \
          $(libgm2_la_M2FLAGS) $< -o $@

DEPFILES=$(libfractapi_la_SOURCES:%.mod=.deps/%.d)
```

The fraction.py wrapper module utilising the fractapi library

```
!/usr/bin/env python3

import fractapi

trace = False

def tprint (*args):
    if trace:
        print (args)

def tprintf (format, *args):
    if trace:
        print(str(format) % args, end="")
```

The fraction.py wrapper module utilising the fractapi library

```
def PI ():
    return fraction ().PI ()

class fraction:
    def __init__ (self, fract = None):
        tprint ("__init__")
        if fract is None:
            tprint ("fract is None")
            fract = fractapi.Constructor (0, 0, 1)
            tprint ("fract now assigned to", fract)
        else:
            tprint ("fract is", fract)
        self._fract = fract
        tprint ("__init__ called", self._fract)
```

The fraction.py wrapper module utilising the fractapi library

```
    def assign (self, whole = 0, numerator = 0, demoninator = 1):
        tprint ("assign")
        self._fract = fractapi.Constructor (whole, numerator, demoninator)
        return self
    def __del__ (self):
        tprint ("__del__ called", self._fract)
        fractapi.Decon (self._fract)
    def __add__ (self, right):
        lid = self._fract
        rid = right._fract
        return fraction (fractapi.Add (lid, rid))
```

The fraction.py wrapper module utilising the fractapi library

```
    def __sub__ (self, right):
        lid = self._fract
        rid = right._fract
        return fraction (fractapi.Sub (lid, rid))
    def __truediv__ (self, right):
        lid = self._fract
        rid = right._fract
        return fraction (fractapi.Div (lid, rid))
    def __mul__ (self, right):
        tprint ("__mul__")
        lid = self._fract
        rid = right._fract
        tprint ("__mul__ calling m2 Mult", lid, rid)
        res = fractapi.Mult (lid, rid)
        tprint ("__mul__ res =", res)
        return fraction (res)
```

The fraction.py wrapper module utilising the fractapi library

```
    def __str__ (self):
        tprint ("print called for", self._fract)
        whole = fractapi.GetWhole (self._fract)
        numerator  = fractapi.GetNumerator (self._fract)
        denominator = fractapi.GetDenominator (self._fract)
        if whole == 0:
            if numerator == 0:
                return "0"
            result = "%d/%d" % (numerator, denominator)
        else:
            result = "%d" % whole
            if numerator != 0:
                result += " "
                value = "%d/%d" % (numerator, denominator)
                result += value
        return result
```

The fraction.py wrapper module utilising the fractapi library

```
    def __repr__ (self):
        tprint ("__repr__ called for", self._fract)
        return "__repr__ called"
    def __eq__ (self, other):
        return fractapi.IsEqual (self._fract, other._fract)
    def PI (self):
        # Assign fraction to PI
        tprint ("assigning PI to fraction")
        self._fract = fractapi.PI ()
        return self
```