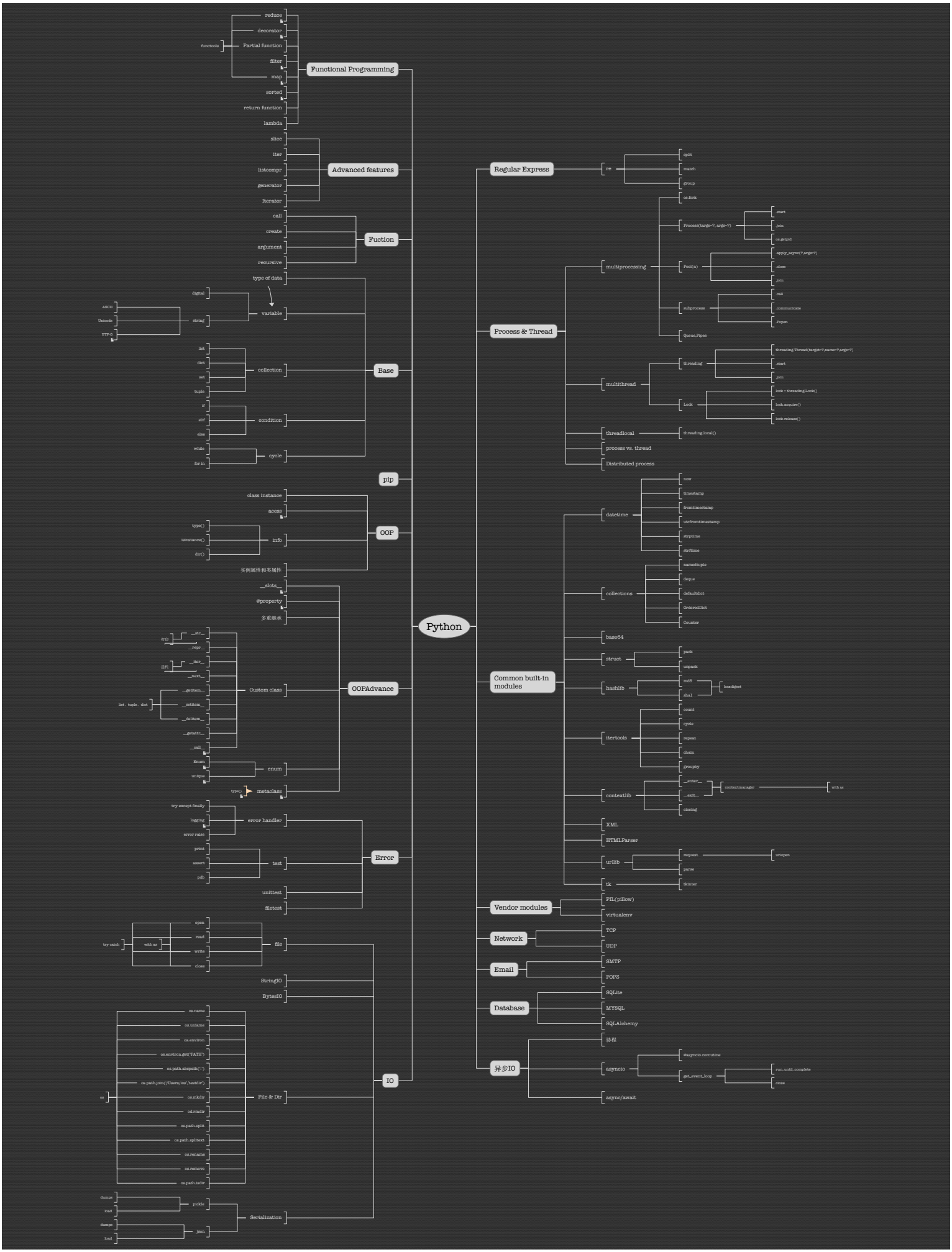
Python
Functional Programming
reduce
decorator
Partial function
functools
filter
map
sorted
return function
lambda
Advanced features
slice
iter
listcompr
generator
Iterator
Fuction
call
create
argument
recursive
Base
type of data
variable
digital
string
ASCII
Unicode
UTF-8
collection
list
dict
set
tuple
condition
if
elif
else
cycle
while
for in
pip
OOP
class instance
acess
info
type()
isinstance()
dir()
实例属性和类属性
OOPAdvance
__slots__
@property
多重继承
Custom class
__str__
打印
__repr__
__iter__
迭代
__next__
__getitem__
__setitem__
__delitem__
__getattr__
__call__
list、tuple、dict
enum
Enum
unique
metaclass
type()
Error
error handler
try except finally
logging
error raise
test
print
assert
pdb
unittest
filetest
IO
file
try catch
with as
open
read
write
close
StringIO
BytesIO
File & Dir
os
os.name
os.uname
os.environ
os.environ.get('PATH')
os.path.abspath('.')
os.path.join('/Users/ios','testdir')
os.mkdir
od.rmdir
os.path.split
os.path.splitext
os.rename
os.remove
os.path.isdir
Serialization
pickle
dumps
load
json
Regular Express
re
split
match
group
Process & Thread
multiprocessing
os.fork
Process(target=?, args=?)
start
join
os.getpid
Pool(n)
apply_async(?,args=?)
close
join
subprocess
call
communicate
Popen
Queue,Pipes
multithread
threading
threading.Thread(target=?,name=?,args=?)
start
join
Lock
lock = threading.Lock()
lock.acquire()
lock.release()
threadlocal
threading.local()
process vs. thread
Distributed process
Common built-in modules
datetime
now
timestamp
fromtimestamp
utcfromtimestamp
strptime
strftime
collections
namedtuple
deque
defaultdict
OrderedDict
Counter
base64
struct
pack
unpack
hashlib
md5
hexdigest
sha1
itertools
count
cycle
repeat
chain
groupby
contextlib
__enter__
__exit__
contextmanager
with as
closing
XML
HTMLParser
urllib
request
urlopen
parse
tk
tkinter
Vendor modules
PIL(pillow)
virtualenv
Network
TCP
UDP
Email
SMTP
POP3
Database
SQLite
MYSQL
SQLAlchemy
异步IO
协程
asyncio
@asyncio.coroutine
get_event_loop
run_until_complete
close
async/await

/Python/Error/error handler/logging

同样是出错，但程序打印完错误信息后会继续执行，并正常退出

/Python/OOPAdvance/__slots__

为了达到限制的目的，Python允许在定义class的时候，定义一个特殊的__slots__变量，来限制该class实例能添加的属性
使用__slots__要注意，__slots__定义的属性仅对当前类实例起作用，对继承的子类是不起作用的

/Python/OOPAdvance/@property

@property的实现比较复杂，我们先考察如何使用。把一个getter方法变成属性，只需要加上@property就可以了，此时，@property本身又创建了另一个装饰器@score.setter，负责把一个setter方法变成属性赋值，于是，我们就拥有一个可控的属性操作

/Python/OOPAdvance/Custom class/__call__

一个对象实例可以有自己的属性和方法，当我们调用实例方法时，我们用instance.method()来调用。能不能直接在实例本身上调用呢?
在Python中，答案是肯定的。 任何类，只需要定义一个__call__()方法，就可以直接对实例进行调用 callable()

/Python/OOPAdvance/enum/Enum

```
from enum import Enum

Month = Enum('Month', ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'))

for name, member in Month.members.items():
print(name, '=>', member, ',', member.value)
```

/Python/OOPAdvance/enum/unique

```
from enum import Enum, unique
@unique
class  Weekday(Enum):
Sun = 0 # Sun的value被设定为0
Mon = 1
Tue = 2
Wed = 3
Thu = 4
Fri = 5
Sat = 6
```

/Python/OOPAdvance/metaclass

metaclass是类的模板，所以必须从`type`类型派生

```
class ListMetaclass(type):
def__new__(cls, name, bases, attrs):
attrs['add'] = lambdaself, value: self.append(value)
returntype.new(cls, name, bases, attrs)

class MyList(list, metaclass=ListMetaclass):
pass
```

/Python/OOPAdvance/metaclass/type()

要创建一个class对象，type()函数依次传入3个参数： class的名称；继承的父类集合，注意Python支持多重继承，如果只有一个父类，别忘了tuple的单元素写法；class的方法名称与函数绑定，这里我们把函数fn绑定到方法名hello上。通过type()函数创建的类和直接写class是完全一样的，因为Python解释器遇到class定义时，仅仅是扫描一下class定义的语法，然后调用type()函数创建出class。

/Python/OOP/acess

如果要让内部属性不被外部访问，可以把属性的名称前加上两个下划线__，在Python中，实例的变量名如果以__开头，就变成了一个私有变量（private），只有内部可以访问，外部不能访问

/Python/Base/variable/string/UTF-8

-- coding: utf-8 --

/Python/Functional Programming/reduce

reduce把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算

/Python/Functional Programming/decorator

本质上，decorator就是一个返回函数的高阶函数。所以，我们要定义一个能打印日志的decorator，可以定义如下：

```
def log(func):
def wrapper(*args, **kw):
print('call %s():'% func.name)
return func(*args, **kw)
return wrapper
```

/Python/Functional Programming/filter

filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素。

/Python/Functional Programming/map

map()函数接收两个参数，一个是函数，一个是Iterable，map将传入的函数依次作用到序列的每个元素，并把结果作为新的Iterator返回

/Python/Functional Programming/sorted

sorted()函数也是一个高阶函数，它还可以接收一个key函数来实现自定义的排序