WELCOME TO PYTHON

MORE THAN BASIC KNOWLEDGE

Wang Xuechen

2018-03-22

AGENDA - 1

- Who is the audiences?
- What is Python, CPython, PyPy, Psyco or JPython?
- What cannot python do?
- Python 2 or Python 3?
- pip & virtualenv & venv
- IDE & Lint & PEP8
- pythonic

AGENDA - 2

- Magic Methods
  - __str__ & __repr__ / __add__ & __sub__ (Operator + -)
  - __new__ & __init__ / __del__
  - __len__ & __getitem__ & __setitem__ & __delitem__
  - __iter__ & next
  - __enter__ & __exit__ & with
  - And so on …… (^ ◇ ^)

AGENDA - 3

- Advanced Features
  - Lambda & List Comprehensions
  - Iterator(Iterable) & Generator / Decorator / Descriptor
- Go Further

AGENDA - EXPANSIONS

- Recursion & Tail recursion optimisation
- GUI Debug Tool
- Closure
- The Zen of Python

WHO IS THE AUDIENCES

- Python beginner, has some basic knowledge.
- To want to know more about Python, but no source.
- No 3rd Python Packages' introduction.
- Boring and Fantastic (￣◇￣;)

WHAT IS PYTHON, CPYTHON, PYPY, PSYCO OR JPYTHON

- Python language and it's Interpreter.
- When people speak of Python they often mean not just the language but also the CPython implementation.
- Python is actually a specification for a language that can be implemented in many different ways.
- Reference
  - The Hitchhiker's Guide to Python!

WHAT CANNOT PYTHON DO

- High-Performance **Compute-Intensive** tasks
  - C/C++ language -> DLL or SO
- **Compute-Intensive** Multithreading (GIL)
  - **I/O-Intensive**
  - Multiprocessing
- GIL (Global Interpreter Lock)
  - Python的GIL是什么鬼，多线程性能究竟如何 @ jobbole.com

PYTHON 2 OR PYTHON 3

- Python 2.x is legacy, Python 3.x is the present and future of the language.
- The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release.
- Reference
    - https://wiki.python.org/moin/Python2orPython3
    - Python 2.7.x 与 Python 3.x 的主要差异 By ShinChan @ github.io

PIP & VIRTUALENV & VENV

- PyPA
    - The Python Packaging Authority (PyPA) is a working group that maintains many of the relevant projects in Python packaging.
- PyPI - the Python Package Index
    - The Python Package Index is a repository of software for the Python programming language. There are currently 132897 packages here.

PIP & VIRTUALENV & VENV

- pip
  - The PyPA recommended tool for installing Python packages.
  - Use pip to install Python packages from PyPI.
- Reference
  - https://www.pypa.io/en/latest/
  - https://pypi.python.org/pypi
  - https://packaging.python.org/guides/tool-recommendations/

PIP & VIRTUALENV & VENV

- Use virtualenv, or venv to isolate application specific dependencies from a shared Python installation.
- Reference
  - https://packaging.python.org/tutorials/installing-packages/#creating-virtual-environments

PIP & VIRTUALENV & VENV

```
python3 -m venv <DIR>
source <DIR>/bin/activate

pip install 'SomeProject>=1,<2'
pip install -r requirements.txt

pip freeze > requirements.txt
```

IDE & LINT & PEP8

- ATOM
  - A hackable text editor for the 21st Century.
  - Packages(plug-in)
  - https://atom.io/
    - https://atom.io/packages/autocomplete-python
    - https://atom.io/packages/linter-flake8
    - https://atom.io/packages/virtualenv
    - https://atom.io/packages/minimap

IDE & LINT & PEP8

- LINT
  - Linting is the process of running a program that will analyse code for potential errors.
  - Reference
    - https://stackoverflow.com/questions/8503559/what-is-linting

IDE & LINT & PEP8

- PEP8
  - Style Guide for Python Code.
  - Reference
    - https://www.python.org/dev/peps/pep-0008/

PYTHONIC

- "Pythonic" means something like "idiomatic Python"
  - a, b = b, a
  - res = ' '.join(strList)
  - l = [x*x for x in range(10) if x % 3 == 0]
  - dic = {'name':'Tim', 'age':23} dic['workage'] = dic.get('workage',0) + 1
  - with
- Reference
  - 让你的Python代码更加pythonic By wuzhiwei
  - Code Like a Pythonista: Idiomatic Python By David Goodger

MAGIC METHODS

- Python Protocol
  - PEP 544 -- Protocols: Structural subtyping (static duck typing)
- Reference
  - Magic Methods and Operator Overloading @ python-course.eu
  - A Guide to Python's Magic Methods @ Rafe Kettler
  - Python 魔术方法指南 @ PyCodersCN

MAGIC METHODS - __STR__ & __REPR__

- __str__(self)
  - Defines behavior for when str() is called on an instance of your class.
- __repr__(self)
  - Defines behavior for when repr() is called on an instance of your class. The major difference between str() and repr() is intended audience. repr() is intended to produce output that is mostly machine-readable (in many cases, it could be valid Python code even), whereas str() is intended to be human-readable.

MAGIC METHODS - __NEW__ & __INIT__

- __new__(cls, [...])
  - __new__ is the first method to get called in an object's instantiation. It takes the class, then any other arguments that it will pass along to __init__. __new__ is used fairly rarely, but it does have its purposes, particularly when subclassing an immutable type like a tuple or a string. I don't want to go in to too much detail on __new__ because it's not too useful, but it is covered in great detail in the Python docs.
- __init__(self, [...])
  - The initializer for the class. It gets passed whatever the primary constructor was called with (so, for example, if we called x = SomeClass(10, 'foo'), __init__ would get passed 10 and 'foo' as arguments. __init__ is almost universally used in Python class definitions.

MAGIC METHODS - __DEL__

- __del__(self)
    - If __new__ and __init__ formed the constructor of the object, __del__ is the destructor. It doesn't implement behavior for the statement del x (so that code would not translate to x.__del__()). Rather, it defines behavior for when an object is garbage collected. It can be quite useful for objects that might require extra cleanup upon deletion, like sockets or file objects. Be careful, however, as there is no guarantee that __del__ will be executed if the object is still alive when the interpreter exits, so __del__ can't serve as a replacement for good coding practices (like always closing a connection when you're done with it. In fact, __del__ should almost never be used because of the precarious circumstances under which it is called; use it with caution!
- Sample code:
    - https://github.com/zenanswer/python-practice/blob/master/cienet/magicmethods/strrepr.py

MAGIC METHODS - __LEN__ & __GETITEM__ & __SETITEM__ & __DELITEM__

- Sample code:
  - https://github.com/zenanswer/python-practice/blob/master/cienet/magicmethods/behindcontainer.py

MAGIC METHODS - __ITER__ & NEXT

- Sample code:
  - https://github.com/zenanswer/python-practice/blob/master/cienet/magicmethods/iternext.py

MAGIC METHODS - __ENTER__ & __EXIT__ & WITH

```
try:
  handle = open something
  handle.do_sth(bababa)

except Exception exp:
  log exp

finally:
  handle.close
```

```
with open something as handle:
    handle.do_sth(bababa)
```

MAGIC METHODS - __ENTER__ & __EXIT__ & WITH

@ 浅谈 Python 的 with 语句 www.ibm.com

```
context_manager = context_expression
exit = type(context_manager).__exit__
value = type(context_manager).__enter__(context_manager)
exc = True   # True 表示正常执行，即便有异常也忽略；False 表示重新抛出异常，需要对异常进行处理
try:
    try:
        target = value  # 如果使用了 as 子句
        with-body     # 执行 with-body
    except:
        # 执行过程中有异常发生
        exc = False
        # 如果 __exit__ 返回 True，则异常被忽略；如果返回 False，则重新抛出异常
        # 由外层代码对异常进行处理
        if not exit(context_manager, *sys.exc_info()):
            raise
finally:
    # 正常退出，或者通过 statement-body 中的 break/continue/return 语句退出
    # 或者忽略异常退出
    if exc:
        exit(context_manager, None, None, None)
    # 缺省返回 None，None 在布尔上下文中看做是 False
```

MAGIC METHODS - __ENTER__ & __EXIT__ & WITH

- Context managers allow setup and cleanup actions to be taken for objects when their creation is wrapped with a with statement. The behavior of the context manager is determined by two magic methods:
  - __enter__(self)
  - __exit__(self, exception_type, exception_value, traceback)

MAGIC METHODS - __ENTER__ & __EXIT__ & WITH

- __enter__(self)
  - Defines what the context manager should do at the beginning of the block created by the with statement. Note that the return value of __enter__ is bound to the target of the with statement, or the name after the as.
- __exit__(self, exception_type, exception_value, traceback)
  - Defines what the context manager should do after its block has been executed (or terminates). It can be used to handle exceptions, perform cleanup, or do something always done immediately after the action in the block. If the block executes successfully, exception_type, exception_value, and traceback will be None. Otherwise, you can choose to handle the exception or let the user handle it; if you want to handle it, make sure __exit__ returns True after all is said and done. If you don't want the exception to be handled by the context manager, just let it happen.

MAGIC METHODS - __ENTER__ & __EXIT__ & WITH

@ 浅谈 Python 的 with 语句 www.ibm.com

```
class DummyResource:
def __init__(self, tag):
        self.tag = tag
        print 'Resource [%s]' % tag
    def __enter__(self):
        print '[Enter %s]: Allocate resource.' % self.tag
        return self   # 可以返回不同的对象
    def __exit__(self, exc_type, exc_value, exc_tb):
        print '[Exit %s]: Free resource.' % self.tag
        if exc_tb is None:
            print '[Exit %s]: Exited without exception.' % self.tag
        else:
            print '[Exit %s]: Exited with exception raised.' % self.tag
            return False   # 可以省略，缺省的None也是被看做是False
```

MAGIC METHODS - __ENTER__ & __EXIT__ & WITH

- __enter__ and __exit__ can be useful for specific classes that have well-defined and common behavior for setup and cleanup. You can also use these methods to create generic context managers that wrap other objects.

ADVANCED FEATURES

- Lambda & List Comprehensions
- Iterator(Iterable) & Generator / Decorator / Descriptor

ADVANCED FEATURES

- Everything is object
- Everything is reference
  - https://github.com/zenanswer/python-practice/blob/master/cienet/descriptor/everythingobject.py

ADVANCED FEATURES - LAMBDA

- filter / map / reduce
  - https://github.com/zenanswer/python-practice/blob/master/src/py/lambda/lambdatest.py
- Reference
  - https://www.python-course.eu/python3_lambda.php

ADVANCED FEATURES - LIST COMPREHENSIONS

- List comprehension is a complete substitute for the lambda function as well as the functions map(), filter() and reduce(). For most people the syntax of list comprehension is easier to be grasped.
- https://github.com/zenanswer/python-practice/blob/master/src/py/lambda/listcomprehensions.py
- Reference
    - https://www.python-course.eu/python3_list_comprehension.php

ADVANCED FEATURES - ITERATOR(ITERABLE)

- Iterable（可迭代对象）
    - __getitem__, return value by index.
        - def __getitem__(self,key)
        - Exception: IndexError
    - __iter__, return a Iterator object for “for” loop.
        - def __iter__(self)

ADVANCED FEATURES - ITERATOR(ITERABLE)

- Iterator（迭代器对象）
  - __next__, return value by an internal index
    - Exception: StopIteration

ADVANCED FEATURES - ITERATOR(ITERABLE)

- Sample code:
  - https://github.com/zenanswer/python-practice/blob/master/cienet/iter/itertest.py
- Reference
  - 对 Python 迭代的深入研究 By Huoty @ github.io

ADVANCED FEATURES - GENERATOR

- yield x
    - https://github.com/zenanswer/python-practice/blob/master/src/py/iterator_generator/frange.py
- y = yield x
    - https://github.com/zenanswer/python-practice/blob/master/src/py/iterator_generator/yieldsend.py
- yield from sub_generator()
    - https://github.com/zenanswer/python-practice/blob/master/src/py/iterator_generator/yieldfrom.py
- Reference
    - https://www.python-course.eu/python3_generators.php

ADVANCED FEATURES - DECORATOR

- What's the function in Python?
  - A variable.

```
>>> def succ(x):
...     return x + 1
...
>>> successor = succ
>>> successor(10)
11
>>> succ(10)
11
```

```
>>> del succ
>>> successor(10)
11
```

- Reference
  - https://www.python-course.eu/python3_decorators.php

ADVANCED FEATURES - DECORATOR

- Functions inside Functions

```
def temperature(t):
    def celsius2fahrenheit(x):
        return 9 * x / 5 + 32

    result = "It's " + str(celsius2fahrenheit(t)) + " degrees!"
    return result

print(temperature(20))
```

ADVANCED FEATURES - DECORATOR

- Functions as Parameters

```
import math

def foo(func):
    print("The function " + func.__name__ + " was passed to foo")
    res = 0
    for x in [1, 2, 2.5]:
        res += func(x)
    return res

print(foo(math.sin))
print(foo(math.cos))
```

ADVANCED FEATURES - DECORATOR

- Functions returning Functions

```
def polynomial_creator(a, b, c):
    def polynomial(x):
        return a * x**2 + b * x + c
    return polynomial

p1 = polynomial_creator(2, 3, -1)
p2 = polynomial_creator(-1, 2, 1)

for x in range(-2, 2, 1):
    print(x, p1(x), p2(x))
```

$$p(x) = a \cdot x^2 + b \cdot x + c$$

ADVANCED FEATURES - DECORATOR

- A Simple Decorator

```
def our_decorator(func):
    def function_wrapper(x):
        print("Before calling " + func.__name__)
        func(x)
        print("After calling " + func.__name__)
    return function_wrapper

def foo(x):
    print("Hi, foo has been called with " + str(x))

print("We call foo before decoration:")
foo("Hi")

print("We now decorate foo with f:")
foo = our_decorator(foo)

print("We call foo after decoration:")
foo(42)
```

ADVANCED FEATURES - DECORATOR

- The Usual Syntax for Decorators in Python

```
def our_decorator(func):
    def function_wrapper(x):
        print("Before calling " + func.__name__)
        res = func(x)
        print(res)
        print("After calling " + func.__name__)
    return function_wrapper

@our_decorator
def succ(n):
    return n + 1

succ(10)
```

```
from random import random, randint, choice

def our_decorator(func):
    def function_wrapper(*args, **kwargs):
        print("Before calling " + func.__name__)
        res = func(*args, **kwargs)
        print(res)
        print("After calling " + func.__name__)
    return function_wrapper

random = our_decorator(random)
randint = our_decorator(randint)
choice = our_decorator(choice)

random()
randint(3, 8)
choice([4, 5, 6])
```

ADVANCED FEATURES - DECORATOR

- Using wraps from functools
  - __name__ (name of the function),
  - __doc__ (the docstring) and
  - __module__ (The module in which the function is defined)

ADVANCED FEATURES - DECORATOR

```
def greeting(func):
    def function_wrapper(x):
        """ function_wrapper of greeting """
        print("Hi, " + func.__name__ + " returns:")
        return func(x)
    function_wrapper.__name__ = func.__name__
    function_wrapper.__doc__ = func.__doc__
    function_wrapper.__module__ = func.__module__
    return function_wrapper
```

```
from functools import wraps

def greeting(func):
    @wraps(func)
    def function_wrapper(x):
        """ function_wrapper of greeting """
        print("Hi, " + func.__name__ + " returns:")
        return func(x)
    return function_wrapper
```

ADVANCED FEATURES - DECORATOR

- Classes instead of Functions
  - The __call__ method
  - Using a Class as a Decorator

```
class decorator2:

    def __init__(self, f):
        self.f = f

    def __call__(self):
        print("Decorating", self.f.__name__)
        self.f()

@decorator2
def foo():
    print("inside foo()")

foo()
```

ADVANCED FEATURES - DESCRIPTOR

- Reference
  - http://pyzh.readthedocs.io/en/latest/Descriptor-HOW-TO-Guide.html
  - https://docs.python.org/3/howto/descriptor.html

ADVANCED FEATURES - DESCRIPTOR

- 默认对属性的访问控制是从对象的字典里面(__dict__)中获取(get), 设置(set)和删除(delete)它。
  - 举例来说， a.x 的查找顺序是, a.__dict__['x'] , 然后 type(a).__dict__['x'] , 然后找 type(a) 的父类(不包括元类(metaclass))。
  - 如果查找到的值是一个描述器, Python就会调用描述器的方法来重写默认的控制行为。这个重写发生在这个查找环节的哪里取决于定义了哪个描述器方法。
  - 注意, 只有在新式类中时描述器才会起作用。(新式类是继承自 type 或者 object 的类)
- https://github.com/zenanswer/python-practice/blob/master/cienet/descriptor/attributeaccess.py

ADVANCED FEATURES - DESCRIPTOR

- Descriptor Protocol
  - descr.__get__(self, obj, type=None) --> value
  - descr.__set__(self, obj, value) --> None
  - descr.__delete__(self, obj) --> None
- 一个对象具有其中任一个方法就会成为描述器，从而在被当作对象属性时重写默认的查找行为。
- 如果一个对象同时定义了 __get__() 和 __set__(),它叫做资料描述器(data descriptor)。仅定义了 __get__() 的描述器叫非资料描述器(常用于方法，当然其他用途也是可以的)。

ADVANCED FEATURES - DESCRIPTOR

- 对于 对象（objects） 来讲：
  - 资料描述器优先于实例变量，实例变量优先于非资料描述器，__getattr__()方法(如果对象中包含的话)具有最低的优先级。
  - type(ins).__dict__['attr'].__get__(ins, type(ins)
- 对于 类（classes） 来讲：
  - Class.__dict__['attr'].__get__(None, Class)
- https://github.com/zenanswer/python-practice/blob/master/cienet/descriptor/mydescriptor.py
- https://github.com/zenanswer/python-practice/blob/master/cienet/descriptor/integer.py

ADVANCED FEATURES - DESCRIPTOR

- @property = Decorator + Descriptor

liaoxuefeng.com

```
class Student(object):

    @property
    def score(self):
        return self._score

    @score.setter
    def score(self, value):
        if not isinstance(value, int):
            raise ValueError('score must be an integer!')
        if value < 0 or value > 100:
            raise ValueError('score must between 0 ~ 100!')
        self._score = value
```

GO FURTHER

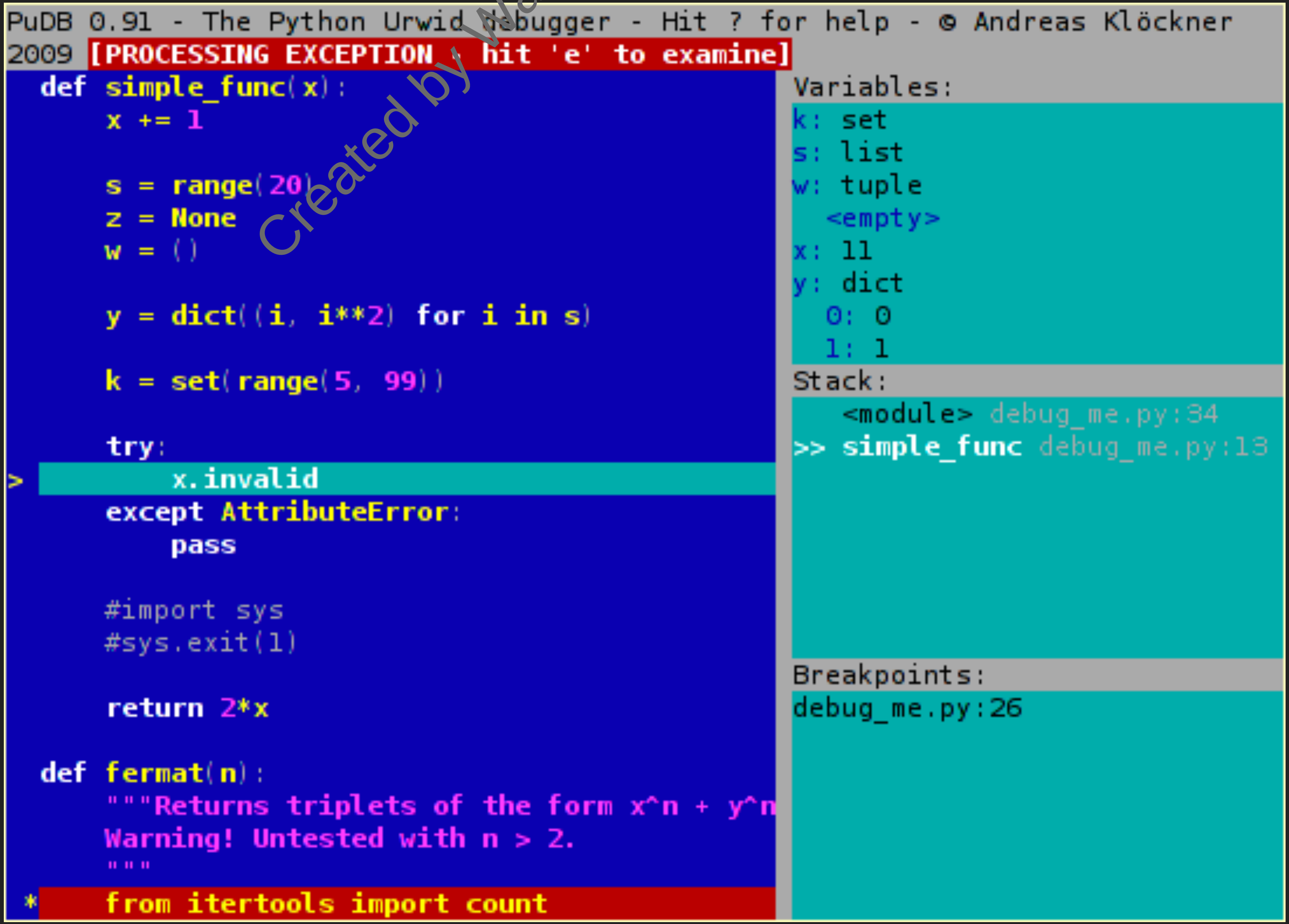
- The Hitchhiker's Guide to Python! (Python最佳实践指南！)
- Python3 Tutorial @ python-course.eu
- Python Cookbook 3rd Edition Documentation (CN)
- Effective python : 编写高质量python代码的59个有效方法

AGENDA - EXPANSIONS(1)

- Recursion & Tail recursion optimisation
  - Python does not support Tail recursion optimisation
    - https://www.quora.com/Why-is-tail-recursion-optimisation-not-implemented-in-languages-like-Python-Ruby-and-Clojure-Is-it-just-difficult-or-impossible
  - Recursion limit: 1000
    - https://stackoverflow.com/questions/3323001/what-is-the-maximum-recursion-depth-in-python-and-how-to-increase-it

AGENDA - EXPANSIONS(2)

- GUI Debug Tool - PuDB
  - pudb @ pypi
  - Documen



```
PuDB 0.91 - The Python Urwid debugger - Hit ? for help - © Andreas Klöckner
2009 [PROCESSING EXCEPTION - hit 'e' to examine]
  def simple_func(x):                          Variables:
      x += 1                                   k: set
                                               s: list
      s = range(20)                            w: tuple
      z = None                                   <empty>
      w = ()                                   x: 11
                                               y: dict
      y = dict((i, i**2) for i in s)             0: 0
                                                 1: 1
      k = set(range(5, 99))                    Stack:
                                                  <module> debug_me.py:34
      try:                                     >> simple_func debug_me.py:13
>         x.invalid
      except AttributeError:
          pass

      #import sys
      #sys.exit(1)
                                               Breakpoints:
      return 2*x                               debug_me.py:26

  def fermat(n):
      """Returns triplets of the form x^n + y^n
      Warning! Untested with n > 2.
      """
 *    from itertools import count
```

AGENDA - EXPANSIONS(3)

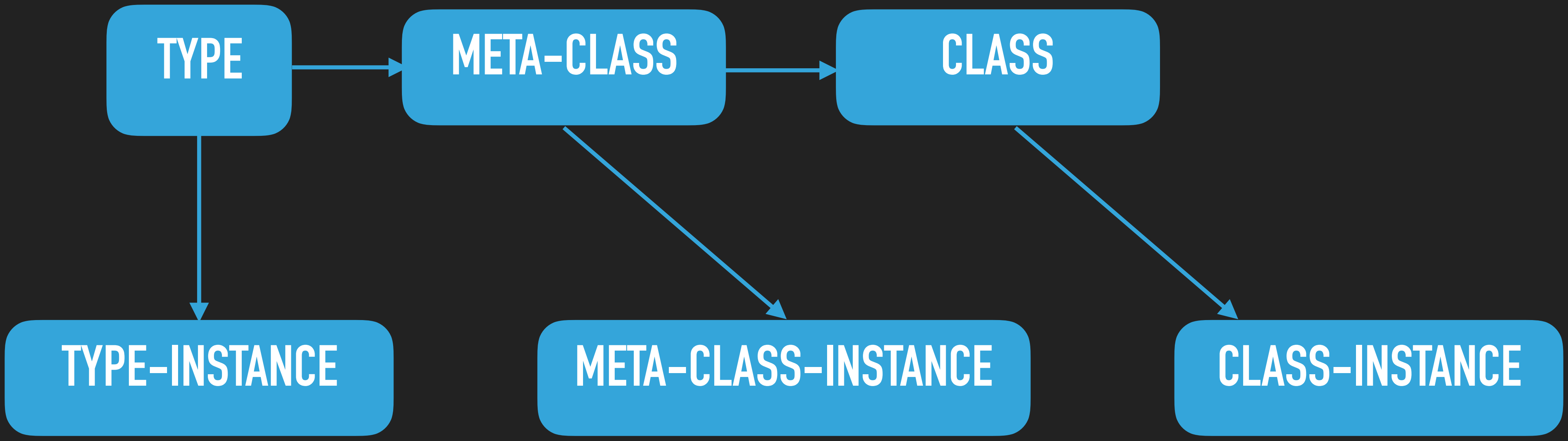
- Closure
  - https://github.com/zenanswer/python-practice/blob/master/cienet/closuretest/my_closure.py

AGENDA – EXPANSIONS(4)

- The Zen of Python
  - PEP 20 -- The Zen of Python

AGENDA - EXPANSIONS(5)

- Meta Class
  - https://github.com/zenanswer/python-practice/tree/master/cienet/meta_class


TYPE
META-CLASS
CLASS
TYPE-INSTANCE
META-CLASS-INSTANCE
CLASS-INSTANCE