General Introduction to Important Python Features

GRK Python Workshop, 14.10.2019

Frank Sauerburger, Manuel Guth

Overview

- New Features in Python 3
- Object-Oriented Programming)
- Generators
- Debugger
- Linter
- argparse
- Packaging
- What not to do

New Features in Python 3

- For all changes have a look at What's New in Python (https://docs.python.org/3/whatsnew/index.html)
- Cheat Sheet: Writing Python 2-3 compatible code (http://python-future.org/compatible_idioms.html)

Sunsetting (https://www.python.org/doc/sunset-python-2/) Python 2

- Python 2.0 was released in 2000
- Python 3.0 in 2006
    - 14 years of parallel support

Support will end on 01.01.2020 (https://pythonclock.org/)

What does that mean?

- No fixes in case of e.g.
    - catastrophic security problems in Python 2
    - bugs in software
- No (fewer) help concerning problems with Python 2

In ATLAS python2.7 is still the default

Print Function

Python 2

```
In [ ]: print "Hello world!"
```

Python 3

```
In [ ]: print("Hello world!")
```

```
In [ ]: print("Hello", "world", sep="-")
```

```
In [ ]: print('home', 'user', 'documents', sep='/')
```

Print Function

In [ ]:

```
print('Mercury', 'Venus', 'Earth', sep=', ', end=", ")
print('Mars', 'Jupiter', 'Saturn', sep=', ', end=', ')
print('Uranus', 'Neptune', 'Pluto', sep=', ')
```

Writing to file

In [ ]:

```
!cat file.txt
```

In [ ]:

```
with open('file.txt', mode='w') as file_object:
    print('hello world', file=file_object)
```

f-Strings

String formatting before Python 3.6

In [ ]:

```
import math
grk = 2_044
where = "HS1"
```

In [ ]:

```
message = "Welcome to the GRK {} Python workshop in {}!\nWe can round pi to {:.2f}".format(grk, where, math.pi)
```

In [ ]:

```
print(message)
```

String formatting with f-String

In [ ]:

```
message_f = f"Welcome to the GRK {grk} Python workshop in {where}!\nWe can round pi to {math.pi:.2f}."
```

In [ ]:

```
print(message_f)
```

True Division

Python 2

3/4 returned 0

Python 3

In [ ]:

```
3/4
```

In python 3 the operator `/` does not loose fractions

Integer division has its own operator

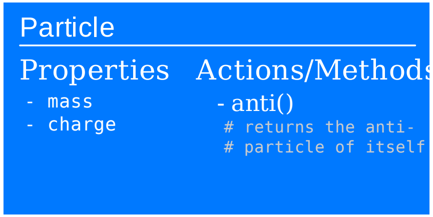
In [ ]:

```
3//4
```

Object-Oriented Programming (OOP)

- Cover only basics
- Partially needed for the rest of the workshop
- No multi-inheritance
- Focused on usage

What is Object-Oriented Programming (OOP)



Particle

Properties
```
- mass
- charge
```

Actions/Methods

- anti()
```
# returns the anti-
# particle of itself
```

- You've used it already:

  ```
  "Hello World".lower()
  ```

  The string `"Hello World"` is an object of `str` class.

- Class is a *blueprint* to create instances, called *objects*
- Combines data and functions
- Example: Particles in an experiment

In [ ]:

```
class Particle:
    def __init__(self, mass, charge):
        self.mass = mass
        self.charge = charge
```

In [ ]:

```
bert = Particle(125, 0)
bert.mass
```

In [ ]:

```
class Particle:
    def __init__(self, mass, charge):
        # __init__() is called when new object is created.
        # First argument (self) is the new object
        self.mass = mass
        self.charge = charge

    def anti(self):
        # First argument is the object on which anti() is called

        # Create new particle with same mass and
        # opposite charge
        return Particle(self.mass, -self.charge)
```

In [ ]:

```
bert = Particle(1.777, -1)
ernie = bert.anti()
ernie.charge
```

In [ ]:

```
ernie.mass
```

In [ ]:

```
bert.charge  # Original particle not changed
```

In [ ]:

```
class Particle:
    def __init__(self, mass, charge):
        # __init__() is called when new object is created.
        # First argument (self) is the new object
        self.mass = mass
        self.charge = charge

    def anti(self):
        # First argument is the object on which anti() is called

        # Create new particle with same mass and
        # opposite charge
        return Particle(self.mass, -self.charge)

    def flip_charge(self):
        # Change the charge of the particle itself (instead of creating a new o
ne)

        self.charge *= -1
```

In [ ]:

```
bert = Particle(1.777, -1)
bert.charge
```

In [ ]:

```
bert.flip_charge()  # Changes the original particle
bert.charge
```

Inheritance

- Sub-classes extend parent classes
- Inheritance models **is a** relationships
    - A `Fermion` **is a** `Particle`
    - A `Particle` is not necessariliy a `Fermion`
- Example: Include sub-classes `Fermion` and `Boson`

In [ ]:

```
class Boson(Particle):
    def interact_with_higgs(self, factor=1.5):
        # Bosons can increase their mass by interacting with the Higgs field (N
EW PHYSICS!)
        self.mass *= factor

class Fermion(Particle):
    def __init__(self, mass, charge, generation):
        super().__init__(mass, charge)  # Create a regular particle
        self.generation = generation
```

In [ ]:

```
tau = Fermion(1.777, -1, 3)
tau.generation
```

In [ ]:

```
Z = Boson(60.78, 0)
Z.mass
```

In [ ]:

```
Z.interact_with_higgs()
Z.mass
```

In [ ]:

```
Z.generation  # Z is a Boson which do not come in generations
```

In [ ]:

```
tau.interact_with_higgs()
```

Other interesting things about OOP

- Override `__str__` and `__repr__` methods
- Override operators: `ernie + bert`
- Polymorphism: Implement methods differently in different sub-classes
    - `Fermion.susy()` returns a Boson
    - `Boson.susy()` returns a Fermion

Exercise: Implement a 2D Vector

Implement a `Vector2D` class such that the following lines work

In [ ]:

```
from solutions import Vector2D
a = Vector2D(4, 3)
a.x
```

In [ ]:

```
a.y
```

In [ ]:

```
a.length()
```

In [ ]:

```
a.scale(3)
a.length()
```

Generators

In [ ]:

```python
def squares(end):
    """
    Returns the squares of 0 up to (not including) the given end.
    >>> squares(3)
    [0, 1, 4]
    """
    out = []
    for i in range(end):
        out.append(i * i)
    return out
```

In [ ]:

```python
squares(3)
```

This is a typical pattern:

1. Create empty list
2. Append items in loop
3. Return final list

Problematic when dealing with huge lists

In [ ]:

```
small_list = squares(10)  # Returns list of 10 items
sum(small_list)
```

In [ ]:

```
large_list = squares(1000_000)  # Returns a list with 1 million items
                                # Calling it with 1 billion exhausts my compute
r's memory
sum(large_list)
```

In this example

- Don't need random access to items: `large_list[100]`
- Need only to iterate over list once

Solution: Generators

In [ ]:

```
def squares(end):
    """
    Returns the squares of 0 up to (not including) the given end.
    >>> squares(3)
    [0, 1, 4]
    """
    # Old implemenation:
    # out = []
    # for i in range(end):
    #     out.append(i * i)
    # return out
    for i in range(end):
        yield i * i  # yield one item at a time
```

In [ ]:

```
squares(3)
```

In [ ]:

```
list(squares(3))
```

In [ ]:

```
sum(squares(1000_000))  # Computes one item at a time
# Works even with 1 billion, takes ~2min
```

Exercise: Write a generator for a binary sequence

The method should take a `limit` parameter. Each item in the sequence is the product of the previous value and $2: a_n = 2 \cdot a_{n-1}$. The sequence starts with 1. The sequence should stop when the `limit` is reached.

In [ ]:

```
from solutions import exp_seq
list(exp_seq(10))
```

In [ ]:

```
sum(exp_seq(10))
```

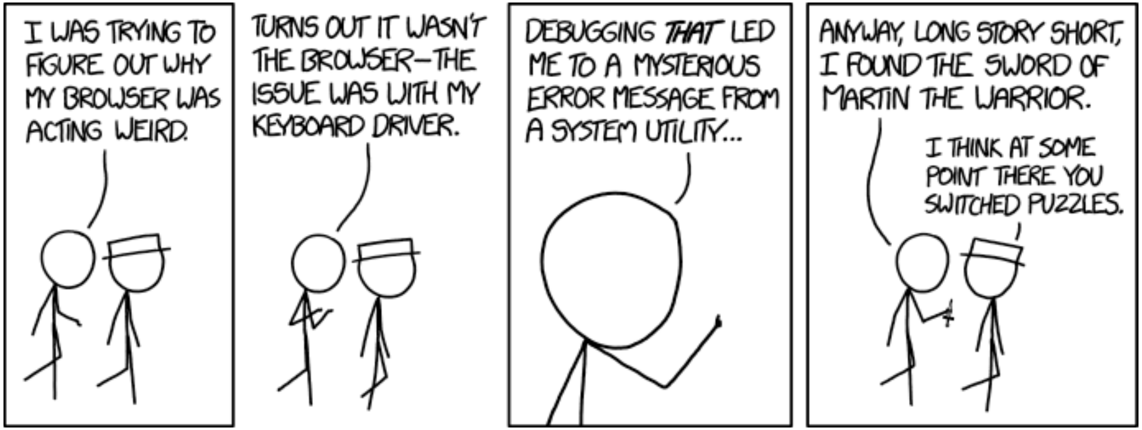
In [ ]:

```
sum(exp_seq(1000_1000))
```

Debugger PDB

Your program crashes or doesn't do what it should?

Debugging can be challenging


I WAS TRYING TO FIGURE OUT WHY MY BROWSER WAS ACTING WEIRD.
TURNS OUT IT WASN'T THE BROWSER—THE ISSUE WAS WITH MY KEYBOARD DRIVER.
DEBUGGING *THAT* LED ME TO A MYSTERIOUS ERROR MESSAGE FROM A SYSTEM UTILITY...
ANYWAY, LONG STORY SHORT, I FOUND THE SWORD OF MARTIN THE WARRIOR.
I THINK AT SOME POINT THERE YOU SWITCHED PUZZLES.

Example

In [ ]:

```
from myproject import read_config, compute_all_results

config = read_config()
# ...
results = compute_all_results(config)  # lengthy computation
# ...
for result in results:
    if result == "tt":
        print("We have the answer!")
        break
else:
    print("This should not happen.")
```

Debugging with `print()`

Add single print, rerun **whole** program

In [ ]:

```
config = read_config()
# ...
results = compute_all_results(config)  # lengthy computation
# ...
print(results)  # Inspect the list of results
for result in results:
    if result == "tt":
        print("We have the answer!")
        break
else:
    print("This should not happen.")
```

- `tt` in results
- Why not detected in loop?

Debugging with `print()`

Add another print, rerun **whole** program **again**

In [ ]:

```
config = read_config()
# ...
results = compute_all_results(config)  # lengthy computation
# ...
print(results)  # Inspect the list of results
for result in results:
    print(result)
    if result == "tt":
        print("We have the answer!")
        break
else:
    print("This should not happen.")
```

Better: Using debugger

Insert `breakpoint()` (or `import pdb; pdb.set_trace()` before Python 3.7) and rerun whole program

In [ ]:

```
config = read_config()
# ...
results = compute_all_results(config)  # lengthy computation
# ...
import pdb; pdb.set_trace()  # This works also before 3.7
for result in results:
    if result == "tt":
        print("We have the answer!")
        break
else:
    print("This should not happen.")
```

Better: Using debugger

- Trigger debugger
  - Add `breakpoint()` or `import pdb; pdb.set_trace()`
  - Run `python -m pdb your_program.py`
- Command summary
  - `b [FILE:]LINE` adds a new **b**earkpoint
  - `c` **c**ontinue to next breakpoint
  - `n` run **n**ext statement
  - `s` **s**tep into method call
  - `u` move one level up (reverts `s`)
  - `cl [N]` clear breakpoints or breakpoint `N`
  - `q` **q**uit
  - `h` **h**elp

Exercise:

Investigate the example below (or online http://cern.ch/go/sb8r (http://cern.ch/go/sb8r)):

In [ ]:

```
cities = set(["London", "Paris", "Bern"])  # Unordered collection

def get_new_cities():
    new_cities = []
    new_cities.append("Oslo")
    new_cities.append("Praque")
    return set(new_cities)

cities.union(get_new_cities())

print(cities)  # Does not include Oslo, Praque!
```

Linter

Scans the code to flag

- Programming errors
- Suspicious constructs ("Code that smells")
- Stylistic errors (Enforces common style within a team)

Several options for Python

- Pylint
- Flake8
- ...

Linter Example

Take the code from the previous exercise.

```python
# debug_exercise.py
cities = set(["London", "Paris", "Bern"])  # Unordered collection

def get_new_cities():
    new_cities = []
    new_cities.append("Oslo")
    new_cities.append("Praque")
    return set(new_cities)

cities.union(get_new_cities())

print(cities)  # Does not include Oslo, Praque!
```

Linter Example

```
$ python -m pylint debug_exercise.py
************* Module debug_exercise
C:  1, 0: Missing module docstring (missing-docstring)
C:  1, 0: Constant name "cities" doesn't conform to UPPER_CASE naming style (i
nvalid-name)
C:  3, 0: Missing function docstring (missing-docstring)

------------------------------------------------------------------
Your code has been rated at 6.25/10 (previous run: 6.25/10, +0.00)
```

- `cities` a constant?
- Would have spotted the mistake already here

Command-line Options - `argparse` (https://docs.python.org/3/library/argparse.html#module-argparse)

Command-line parsing module in the Python standard library

In [ ]:

```
from argparse import ArgumentParser
```

In [ ]:

```
parser = ArgumentParser()
```

In [ ]:

```
parser.add_argument("number", type=float) # positional argument with type float
```

In [ ]:

```
parser.add_argument('-e', '--exponent', default=2, type=int) # option with default value and int type
```

In [ ]:

```
parser.add_argument("-v", "--verbose", help="increase output verbosity",
                       action="store_true") # true/false option with help message
```

Command-line Options - argparse (https://docs.python.org/3/library/argparse.html#module-argparse)

In [ ]:

```
%%writefile argparse_test.py
from argparse import ArgumentParser

parser = ArgumentParser()
parser.add_argument("number", type=float) # positional argument with type float

parser.add_argument('-e', '--exponent', default=2, type=int) # option with default value and int type

parser.add_argument("-v", "--verbose", help="increase output verbosity",
                    action="store_true") # true/false option with help message
args = parser.parse_args()

if args.verbose is True:
    print(f"{args.number}^{args.exponent} =", args.number ** args.exponent)
else:
    print(args.number ** args.exponent)
```

In [ ]:

```
!python argparse_test.py -h
```

Command-line Options - `argparse` (https://docs.python.org/3/library/argparse.html#module-argparse)

Alternatives:

- `click` (http://click.pocoo.org/6/)
- `docopt` (http://docopt.org/)

Exercise: Hello World Argparse

Write a python script `hello_world.py` with a language option (de, en, fr, etc.) and the name.

The default should be language=en and name=World

which should return e.g.

```
python hello_world.py -l de --name Bert
>> Hallo Bert
```

```
python hello_world.py
>> Hello World
```

Solution: Hello World Argparse

In [ ]:

```
%%writefile argparse_exercise.py
from argparse import ArgumentParser

greetings = {"en": "Hello", "de": "Hallo", "fr": "Salut", "it": "Buongiorno"}

parser = ArgumentParser()
parser.add_argument('-l', '--language', default="en", type=str, choices=list(greetings.keys()))
parser.add_argument("-n", "--name", default="World", type=str)
args = parser.parse_args()

print(greetings[args.language], args.name)
```

In [ ]:

```
!python argparse_exercise.py -l it -n Bert
```

Packaging

- Split larger projects into modules and packages
- `setuptools`/`disttools` allows combining packages, scripts and metadata
- Easily shared with other people
- **Task: Build package from the `hello_world.py` example and share it with other people**

Terminology

Module

- Single `.py` file
- Usable via `import FILE_NAME` from file in same directory (directory has to be in `sys.path`)

Package

- Method to structure Python namepsace (e.g. `os.path.join`)
- Created with `PACKAGE_NAME/__init__.py` file
- `__init__.py` could be empty
- Package directory can host modules and packages
- Usable via `import PACKAGE_NAME` from parent directory (directory has to be in `sys.path`)

Packaging: 1. Create package

We will call the package `hellolib` Create the file `hellolib/__init__.py` . *Reuse the code from the previous exercise.*

In [ ]:

```
%%writefile hellolib/__init__.py
"""
The Python module hellolib/__init__.py hosts a method to greet the world.
"""

def print_greeting(greeting, name):
    """
    Prints the custom greeting.
    """
    print("%s %s!" % (greeting, name))
```

Method available via

In [ ]:

```
import hellolib
hellolib.print_greeting("Bonjour", "Ernie")
```

Packaging: 2. Add command-line tool

We want to include the command-line tool with argparse. Create `scripts/hello_world`.

Reuse the code from the previous exercise.

In [ ]:

```
%%writefile scripts/hello_world
#! /usr/bin/env python3
# Change path for windows

import argparse
from hellolib import print_greeting

greetings = {"en": "Hello", "de": "Hallo", "fr": "Salut", "it": "Buongiorno"}

parser = argparse.ArgumentParser()
parser.add_argument("-n", "--name", default="World")
parser.add_argument("-l", "--lang", choices=greetings)
args = parser.parse_args()

print_greeting(greetings[args.lang], args.name)
```

Packaging: 3. Add metadata with setuptools

Create a `setup.py` alongside the directories `scripts/` and `hellolib/`.

In [ ]:

```
%%writefile setup.py
from setuptools import setup, find_packages
setup(
    name="hellolib",
    version="0.0.1",
    packages=find_packages(),
    scripts=['scripts/hello_world'],
    install_requires=[],  # We don't have any dependencies
    author="Me",
    author_email="me@example.com",
    # Much more: https://setuptools.readthedocs.io/
)
```

Packaging: 4. Install or share

- Run `python3 setup.py install` to copy files
    - `import hellolib` from any directory
    - Run `hello_world` command in any directory
- Run `python3 setup.py develop` to link instead of copy
    - Changes are propagated on your system
- Share the directory with others: Email/Download zip, Git repository
- Use `twine` to publish your package on `pypi.org`

What NOT to do

Thinks you should avoid with python

Misusing default arguments in functions

you can define default values in a function

In [ ]:

```
def grk_append(grk_list=[]):  # grk_list is optional with the default value []
    grk_list.append("grk") # this line can cause problems!
    return grk_list
```

In [ ]:

```
grk_append()
```

Possible way out of it

In [ ]:

```
def grk_append(grk_list=None):  # setting default value to None
    if grk_list is None:
        grk_list = []
    grk_list.append("grk")
    return grk_list
```

In [ ]:

```
grk_append()
```

Import Mistakes

Wildcard Import

In [ ]:

```
from numpy import *
```

- Can cause name clashing
- Unnecessary import of unneeded functionalities

with python 3 e.g. ROOT does not allow wildcard import anymore

```
from ROOT import *
```

Import Mistakes

Name conflicts with other libraries

email is a python standard library

```
from email.message import EmailMessage
```

```
%%writefile email.py
def GetMail():
    return "grk@physik.uni-freiburg.de"
```

```
import email
email.GetMail()
```

Opening files

Often used to open files

```
file = open("test.txt", "w")
.
.
.
file.close()
```

This synthax can cause issues e.g. if there is an exception raised before `file.close()`

Saver way to open files

```
with open("test.txt", "w") as file:
    .
    .
    .
```

Mutable assignment errors - Dictionaries

We have a dictionary a

In [ ]:

```
a = {'1': "one", '2': 'two'}
```

Now we want to have the same dict again but leaving the previous one intact

In [ ]:

```
b = a
```

In [ ]:

```
b
```

In [ ]:

```
b['3'] = "three"
```

In [ ]:

```
a
```

Mutable assignment errors - Dictionaries

What happened?

Here b is a pointer -> reference to a.

The same thing is happening for lists.

Possible way out:

In [ ]:

```
# for dicts
b = a.copy()
# for lists
l = list(a.keys())
cp = l[:]
```

In [ ]: