HIGH-PERFORMANCE Python

PART 2: SINGLE THREAD

High-performance Python

https://github.com/squillero/python-lectures/

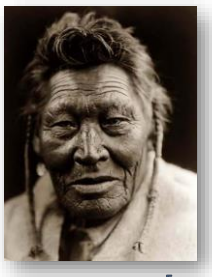
september 2023 **version 0.5**

Why HP Python?

- Python is not quite fast, at least not compared to C++ or Rust
- … but it's not quite slow either
  - Large projects
  - System language
  - High-level algorithms
- Goal of HP Python
  - Avoid pitfalls
  - High-level optimization
  - Pure-Python optimization

Rule of Optimization (personal)

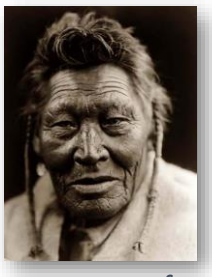
- Don't write it
- Don't do it
- Exploit parallelism



Rule of Optimization (personal)

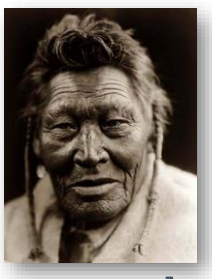
- Don't write it
  - Use builtins
  - Use standard libraries
- Don't do it
- Exploit parallelism

Lecture 1



Rule of Optimization (personal)

- Don't write it
  - Use builtins
  - Use standard libraries
- Don't do it
  - Don't do it yet: i.e., use lazy execution
  - Don't do it again: i.e., cache previous calls
- Exploit parallelism

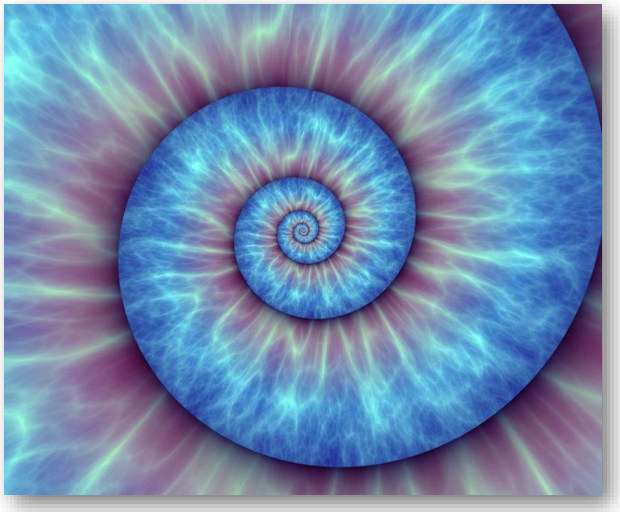


Lazy Executions & Generators

```
def number_generator(n):
    i = 0
    while i < n:
        yield i
        i += 1

for number in number_generator(5):
    print(number)
```


Problem: Write a Generator for Fibonacci



Generators & Lazy Evaluation

- `any()`
- `all()`

Problem: Calculate Prime Numbers

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Out-of-the-box

New in version 3.9

```
import functools
```

```
@functools.cache
def cached_function(num):
    print(f"Performing costly calculation for {num}")
    return num * 2
```

```
cached_function(42)
```

```
Performing costly calculation for 42
84
```

```
cached_function(42)
```

```
84
```

```
cached_function(43)
```

```
Performing costly calculation for 43
86
```

Out-of-the-box

```python
@functools.lru_cache(maxsize=512)
def cached_function(num):
    print(f"Performing costly calculation for {num}")
    return num * 2
```

```python
cached_function(42)
```

```
Performing costly calculation for 42
84
```

```python
cached_function.cache_info()
```

```
CacheInfo(hits=0, misses=1, maxsize=128, currsize=1)
```

```python
cached_function.cache_clear()
```

```python
cached_function(42)
```

```
Performing costly calculation for 42
84
```


Out-of-the-box (caveats)

- **`@functools.cached_property`**
  - Similar to property, but with caching
- **`@functools.lru_cache(typed)`**
  - If **`typed`** is **`True`**, function arguments of different types will be cached separately
  - If **`typed`** is **`False`** (default), the implementation will ***"usually"*** regard different types as equivalent calls and only cache a single result — but some types ***"may"*** be always cached separately

Memoization: joblib

```
from joblib import Memory
memory = Memory('./cache', verbose=0)
```

```
@memory.cache
def memoized_function(num):
    print(f"Performing even more costly calculation for {num}")
    return num * 3
```

```
memoized_function(13)
```

```
Performing even more costly calculation for 13
39
```

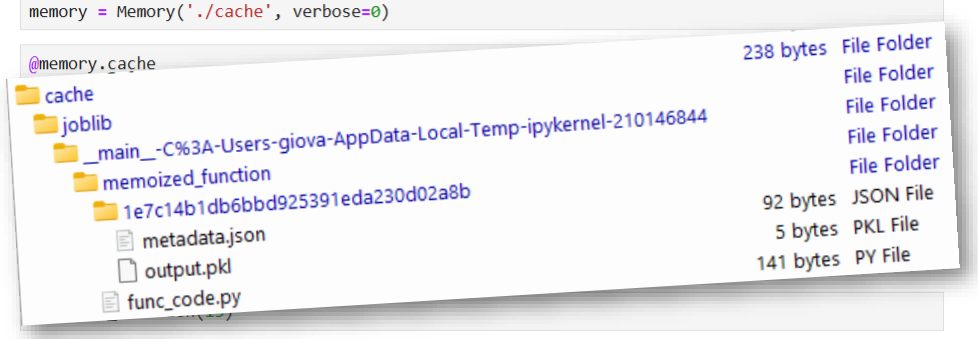
```
memoized_function(13)
```

```
39
```


Memoization: joblib

```
from joblib import Memory
memory = Memory('./cache', verbose=0)
```

```
@memory.cache
```




```
39
```

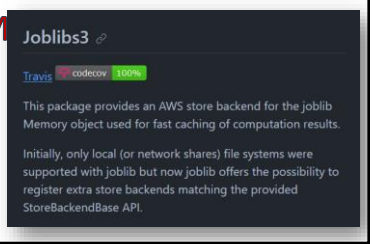
Memoization: joblib

```
Memory(
    – location=None          // path/base directory
    – backend='local'        // local is regular file system
    – mmap_mode=None         // see numpy.memmap
    – compress=False         // compressed array cannot use memmap
    – verbose=1              // ...
    – bytes_limit=None       // int or string like "16M"
    – backend_options=None
)
```


Memoization: joblib


```
_STORE_BACKENDS = {'local': FileSystemStoreBackend}
```


```
Mem
    – backend='local'        // local is regular file system
    – mmap_mod  one          // see numpy.memmap
    – compress=  lse         // compressed array cannot use memmap
      bose=1                 // ...
    –  es_limit=None         // int or string like "16M
    – backend_options=None
)
```



Memoization: joblib

```
from joblib import Mem
memory = Memory('./cach      bose=0)
```

```
@memory.cache(ignore=['foo'], verbose=2)
def memoized_function(num, foo):
    print(f"Performing even more costly calculation for {num}")
    return num * 3
```

```
memoized_function(13, foo=1)
```

```
_____
[Memory] Calling __main__-C%3A-Users-giova-AppData-Local-Temp-ipykernel-499165028.memoized_function...
memoized_function(13, foo=1)
Performing even more costly calculation for 13
_____memoized_function - 0.0s, 0.0min
39
```

```
memoized_function(13, foo=2)
```

```
[Memory]776.5s, 12.9min : Loading memoized_function...
39
```


Memoization: joblib

```
def my_function(*args, **kwargs):
    print(args, kwargs)
    return True

@memory.cache(cache_validation_callback=my_function, verbose=2)
def memoized_function(num):
    print(f"Performing even more costly calculation for {num}")
    return num * 3
```

```
memoized_function(13)
```

```
_____
[Memory] Calling __main__-C%3A-Users-giova-AppData-Local-Temp-ipykernel-56899765.memoized_function...
memoized_function(13)
Performing even more costly calculation for 13
_____memoized_function - 0.0s, 0.0min
39
```

```
memoized_function(13)
```

```
({'duration': 0.001997709274291992, 'input_args': {'num': '13'}, 'time': 1697901174.566357},) {}
[Memory]41.1s, 0.7min   : Loading memoized_function...
39
```

Generic Persistence: `pickle`

```
import pickle
```

```
cache = dict()
def cached_function(num):
    if num not in cache:
        print(f"Perfoming costly calculation for {num}")
        cache[num] = num * 3 + 1
    return cache[num]
```

```
cached_function(42)
```

```
Perfoming costly calculation for 42
127
```

```
cached_function(42)
```

```
127
```

```
pickle.dump(cache, open('cache.pkl', 'wb'))
```

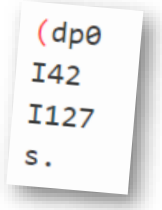
```
cache = pickle.load(open('cache.pkl', 'rb'))
```

```
cached_function(42)
```

```
127
```


Generic Persistence: `pickle`

- Supports dumping to/loading from a string (`dumps`/`loads`)
- Supports different *protocols*
  - `pickle.DEFAULT_PROTOCOL`
  - `0` (zero) for almost human-readable data
  - `pickle.HIGHEST_PROTOCOL`



Generic Persistence: `pickle`

`pickle` vs. `joblib`

```python
import joblib
```

```python
joblib.dump(cache, open('cache.pkl', 'wb'))
```

```python
cache = joblib.load(open('cache.pkl', 'rb'))
```

```python
cached_function(42)
```

```
127
```

Note:

As of Python 3.8 and numpy 1.16, pickle protocol 5 introduced in PEP 574 supports efficient serialization and de-serialization for large data buffers natively using the standard library:

```
pickle.dump(large_object, fileobj, protocol=5)
```

`pickle` vs. `marshal`

- Python has a more primitive serialization module called **`marshal`**, but in general pickle should always be the preferred way to serialize Python objects
- **`marshal`** exists primarily to support Python's **`.pyc`** files

Generic Persistence: `shelve`

```
import shelve
```

```
cache = None
def shelved_function(num):
    key = str(num)
    if key not in cache:
        print(f"Perfoming costly calculation for {num}")
        cache[key] = num * 3 + 1
    return cache[key]
```

```
with shelve.open('cache.db') as cache:
    print(shelved_function(42))
    print(shelved_function(42))
```

```
Perfoming costly calculation for 42
127
127
```

```
with shelve.open('cache.db') as cache:
    print(shelved_function(42))
    print(shelved_function(42))
```

```
127
127
```

Generic Persistence: `shelve`

```
import shelve

cache = None
def shelved_functio
    key = str(num)
    if key not in cache:
        print(f"Perfoming costly calculati
        cache[key] = num * 3 + 1
    return cache[key]

with shelve.open('cache.db'
    print(shelved_function(42
    print(shelved_function(42

Perfoming costly calculation fo
127
127

with shelve.open('cache.db') as cache:
```

The **keys** in a shelf are ordinary strings

The **values** can be anything that the pickle module can handle, including most class instances, recursive data types, and objects containing lots of shared sub-objects

cache.db.bak	14 bytes	BAK File
cache.db.dat	5 bytes	DAT File
cache.db.dir	14 bytes	DIR File

Bonus Topic: `weakref`

- A weak reference is not enough to keep an object alive
  - When the only remaining references to a referent are weak references, garbage collection may destroy the referent
- Note: Until the object is actually destroyed the weak reference may return the object

```
from dataclasses import dataclass
@dataclass
class Foo:
    num: int

x = Foo(23)
y = weakref.ref(x)

x, y, y()

(Foo(num=23), <weakref at 0x1067b9c60; to 'Foo' at 0x105f88390>, Foo(num=23))
```

Bonus Topic: weakref.proxy

```
from dataclasses import dataclass
import weakref
```

```
@dataclass
class Foo:
    num: int
```

```
store = [Foo(n) for n in range(5)]
refs = [weakref.proxy(store[n]) for n in range(5)]
```

```
refs
```

```
[<weakproxy at 0x106070590 to Foo at 0x1068eb010>,
 <weakproxy at 0x106b5cc70 to Foo at 0x106076610>,
 <weakproxy at 0x106d50220 to Foo at 0x106077190>,
 <weakproxy at 0x106d513a0 to Foo at 0x106900fd0>,
 <weakproxy at 0x106d51530 to Foo at 0x106900a10>]
```

```
del store[2]
refs
```

```
[<weakproxy at 0x106070590 to Foo at 0x1068eb010>,
 <weakproxy at 0x106b5cc70 to Foo at 0x106076610>,
 <weakproxy at 0x106d50220 to NoneType at 0x103a65d48>,
 <weakproxy at 0x106d513a0 to Foo at 0x106900fd0>,
 <weakproxy at 0x106d51530 to Foo at 0x106900a10>]
```


Bonus Topic: weakref.proxy

```
from dataclasses import dataclass
import weakref
```

```
@dataclass
class Foo:
    num: int
```

```
store = [Foo(n) for n in range(5)]
refs = [weakref.proxy(store[n]) for n in range(5)]
```

```
refs
```

```
[<weakproxy at 0x106070590 to Foo at 0x1068eb010>,
 <weakproxy at 0x106b5cc70 to Foo at 0x106076610>,
 <weakproxy at 0x106d50220 to Foo at 0x106077190>,
 <weakproxy at 0x106d513a0 to Foo at 0x106900fd0>,
 <weakproxy at 0x106d51530 to Foo at 0x106900a10>]
```

```
del store[2]
refs
```

```
[<weakproxy at 0x106070590 to Foo at 0x1068eb010>,
 <weakproxy at 0x106b5cc70 to Foo at 0x106076610>,
 <weakproxy at 0x106d50220 to NoneType at 0x103a65d48>,
 <weakproxy at 0x106d513a0 to Foo at 0x106900fd0>,
 <weakproxy at 0x106d51530 to Foo at 0x106900a10>]
```

giovanni.squillero@polito.it High-performance Python 31

Bonus Topic: `weak` Dictionaries

- **`WeakKeyDictionary`**
  - Mapping class that references keys weakly
- **`WeakValueDictionary`**
  - Mapping class that references values weakly
- **`keyrefs`** & **`valuerefs`**
- **`WeakSet`**
  - Set class that keeps weak references to its elements

More Generic Persistence

- **`dbm`**
  - Interfaces to the traditional NoSQL "databases" available under Unix (i.e., fast, single-keyed, key-value store)
- **`sqlite3`**
  - DB-API 2.0 interface for SQLite


Google
squillero
squillero
squillero polito
squillero giovanni
squillero github
Google 搜索
手气不错
giovanni.squillero@polito.it