lecture_20

October 16, 2022

1 Lecture 20

[1]:
```
!python --version
```

```
Python 3.9.13
```

[3]:
```
def fibo(n):
    if n == 0 or n == 1:
        return 1
    return fibo(n-1) + fibo(n-2)
```

[6]:
```
fibo(42)
```

[6]:
```
433494437
```

1.1 Memoization

[9]:
```
fibo_values = {
    0: 1,
    1: 1,
}
```

[10]:
```
def fibo(n):
    if n in fibo_values:
        print(f"Value for {n} already in fibo_values")
        return fibo_values[n]
    print(f"Computing value for {n}")
    val = fibo(n-1) + fibo(n-2)
    fibo_values[n] = val
    return val
```

[11]:
```
fibo(42)
```

```
Computing value for 42
Computing value for 41
Computing value for 40
Computing value for 39
Computing value for 38
```

```
Computing value for 37
Computing value for 36
Computing value for 35
Computing value for 34
Computing value for 33
Computing value for 32
Computing value for 31
Computing value for 30
Computing value for 29
Computing value for 28
Computing value for 27
Computing value for 26
Computing value for 25
Computing value for 24
Computing value for 23
Computing value for 22
Computing value for 21
Computing value for 20
Computing value for 19
Computing value for 18
Computing value for 17
Computing value for 16
Computing value for 15
Computing value for 14
Computing value for 13
Computing value for 12
Computing value for 11
Computing value for 10
Computing value for 9
Computing value for 8
Computing value for 7
Computing value for 6
Computing value for 5
Computing value for 4
Computing value for 3
Computing value for 2
Value for 1 already in fibo_values
Value for 0 already in fibo_values
Value for 1 already in fibo_values
Value for 2 already in fibo_values
Value for 3 already in fibo_values
Value for 4 already in fibo_values
Value for 5 already in fibo_values
Value for 6 already in fibo_values
Value for 7 already in fibo_values
Value for 8 already in fibo_values
Value for 9 already in fibo_values
Value for 10 already in fibo_values
```

```
Value for 11 already in fibo_values
Value for 12 already in fibo_values
Value for 13 already in fibo_values
Value for 14 already in fibo_values
Value for 15 already in fibo_values
Value for 16 already in fibo_values
Value for 17 already in fibo_values
Value for 18 already in fibo_values
Value for 19 already in fibo_values
Value for 20 already in fibo_values
Value for 21 already in fibo_values
Value for 22 already in fibo_values
Value for 23 already in fibo_values
Value for 24 already in fibo_values
Value for 25 already in fibo_values
Value for 26 already in fibo_values
Value for 27 already in fibo_values
Value for 28 already in fibo_values
Value for 29 already in fibo_values
Value for 30 already in fibo_values
Value for 31 already in fibo_values
Value for 32 already in fibo_values
Value for 33 already in fibo_values
Value for 34 already in fibo_values
Value for 35 already in fibo_values
Value for 36 already in fibo_values
Value for 37 already in fibo_values
Value for 38 already in fibo_values
Value for 39 already in fibo_values
Value for 40 already in fibo_values
```

[11]:
```
433494437
```

[15]:
```
def fibo_slow(n):
    if n == 0 or n == 1:
        return 1
    return fibo_slow(n-1) + fibo_slow(n-2)
```

[6]:
```
from timeit import timeit
```

[27]:
```
timeit("fibo_slow(24)", setup="from __main__ import fibo_slow", number=10)
```

[27]:
```
0.13193262499999037
```

[28]:
```
fibo_values = {
    0: 1,
    1: 1,
```

```
}
```

[29]:
```
def fibo(n):
    if n in fibo_values:
        return fibo_values[n]
    val = fibo(n-1) + fibo(n-2)
    fibo_values[n] = val
    return val
```

[30]:
```
timeit("fibo(24)", setup="from __main__ import fibo", number=10)
```

[30]:
```
2.479199997651449e-05
```

[31]:
```
timeit("fibo_slow(42)", setup="from __main__ import fibo_slow", number=1)
```

[31]:
```
58.88020933300004
```

[32]:
```
fibo_values = {
    0: 1,
    1: 1,
}
timeit("fibo(42)", setup="from __main__ import fibo", number=10)
```

[32]:
```
1.2749999996231054e-05
```

[34]:
```
2**41
```

[34]:
```
2199023255552
```

[20]:
```
import sys
```

[21]:
```
sys.getrecursionlimit()
```

[21]:
```
13000
```

[28]:
```
sys.setrecursionlimit(50000)
```

[29]:
```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

[30]:
```
timeit("factorial(12000)", setup="from __main__ import factorial", number=100)
```

[30]:
```
4.930418582999991
```

[31]:
```
from functools import cache
```

[32]:
```
@cache
def factorial_cached(n):
    if n == 0:
        return 1
    return n * factorial_cached(n - 1)
```

[33]:
```
timeit("factorial_cached(12000)", setup="from __main__ import␣
 ↪factorial_cached", number=100)
```

[33]: 0.07344387499999527

[36]:
```
@cache
def fibo_slow_cached(n):
    if n == 0 or n == 1:
        return 1
    return fibo_slow_cached(n-1) + fibo_slow_cached(n-2)
```

[37]:
```
timeit("fibo_slow_cached(42)", setup="from __main__ import fibo_slow_cached",␣
 ↪number=100)
```

[37]: 0.00042441700000495075

[62]:
```
from collections import defaultdict
```

[63]:
```
d = {}
```

[64]:
```
d['a'] = 42
```

[65]:
```
d
```

[65]: {'a': 42}

[66]:
```
d['b'] = {}
```

[67]:
```
d['b']['key'] = 'val'
```

[68]:
```
d
```

[68]: {'a': 42, 'b': {'key': 'val'}}

[69]:
```
d['c']['key'] = 'val'
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [69], in <cell line: 1>()
----> 1 d['c']['key'] = 'val'
```

```
KeyError: 'c'
```

[70]:
```
d = defaultdict(dict)
```

[71]:
```
d
```

[71]:
```
defaultdict(dict, {})
```

[72]:
```
d['a']['key'] = 'value'
```

[73]:
```
d
```

[73]:
```
defaultdict(dict, {'a': {'key': 'value'}})
```

[74]:
```
d['c']
```

[74]:
```
{}
```

[83]:
```
from functools import wraps
```

[84]:
```
def our_cache(func):
    cached_values = defaultdict(dict)
    @wraps(func)
    def wrapper(*args, **kwargs):
        if (args, str(kwargs)) in cached_values[func]:
            return cached_values[func][(args, str(kwargs))]
        res = func(*args, **kwargs)
        cached_values[func][(args, str(kwargs))] = res
        return res
    return wrapper
```

[85]:
```
@our_cache
def factorial_our_cached(n):
    if n == 0:
        return 1
    return n * factorial_our_cached(n - 1)
```

[86]:
```
factorial_our_cached(10)
```

[86]:
```
3628800
```

[87]:
```
timeit("factorial_our_cached(12000)", setup="from __main__ import␣
 ↪factorial_our_cached", number=100)
```

[87]:
```
0.14388662500005012
```

[88]:
```
@our_cache
def fibo_slow_our_cached(n):
    if n == 0 or n == 1:
        return 1
    return fibo_slow_our_cached(n-1) + fibo_slow_our_cached(n-2)
```

[89]:
```
fibo_slow_our_cached(42)
```

[89]:
```
433494437
```

[90]:
```
timeit("fibo_slow_our_cached(42)", setup="from __main__ import␣
 ↪fibo_slow_our_cached", number=100)
```

[90]:
```
0.0002908329997808323
```

[91]:
```
factorial_our_cached.__name__
```

[91]:
```
'factorial_our_cached'
```

[92]:
```
from functools import cached_property
```

[96]:
```
class Employee:
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname

    @cached_property
    def email(self):
        return f"{self.name}.{self.surname}@aca.am".lower()
```

[97]:
```
employee_1 = Employee("Adam", "Smith")
```

[98]:
```
employee_1.email
```

[98]:
```
'adam.smith@aca.am'
```

[99]:
```
employee_1.name = "Jack"
```

[100]:
```
employee_1.email
```

[100]:
```
'adam.smith@aca.am'
```

[101]:
```
domain = "aca.am"


@cache
def generate_email(name, surname):
```

```
    return f'{name}.{surname}@{domain}'
```

[102]:
```
generate_email("Henry", "Harutyunyan")
```

[102]:
```
'Henry.Harutyunyan@aca.am'
```

[103]:
```
domain = "google.com"
```

[104]:
```
generate_email("Henry", "Harutyunyan")
```

[104]:
```
'Henry.Harutyunyan@aca.am'
```

[110]:
```
class Foo:
    def __init__(self, name):
        self.name = name
```

[111]:
```
foo = Foo("adam")
```

[112]:
```
foo.name
```

[112]:
```
'adam'
```

[113]:
```
foo.name = "jack"
```

[115]:
```
foo.name
```

[115]:
```
'jack'
```

[126]:
```
class Foo:
    def __init__(self, name):
        self.__name = name

    @property
    def name(self):
        return self.__name
```

[127]:
```
foo = Foo("Adam")
```

[128]:
```
foo.name
```

[128]:
```
'Adam'
```

[129]:
```
foo.name = "Jack"
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [129], in <cell line: 1>()
```

```
----> 1 foo.name = "Jack"

AttributeError: can't set attribute
```

[136]:
```
class Rectangle:
    def __init__(self, length, width):
        self.__length = length
        self.__width = width

    @property
    def length(self):
        return self.__length

    @property
    def width(self):
        return self.__width

    @cached_property
    def __area(self):  # 20 * 2 = 40
        # logic that takes time (2 seconds)
        return self.length * self.width
```

[137]:
```
rectangle = Rectangle(10, 20)
```

[138]:
```
rectangle.area
```

[138]: 200

[139]:
```
rectangle.length = 15
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [139], in <cell line: 1>()
----> 1 rectangle.length = 15

AttributeError: can't set attribute
```

[140]:
```
rectangle.area
```

[140]: 200

[141]:
```
def foo(x):
    return x**2
```

[142]:
```
a = [1, 2, 3, 4]
```

[143]:
```
for i in map(foo, a):
    print(i)
```

```
1
4
9
16
```

[144]:
```
b = [5, 6, 7, 8]
```

[145]:
```
for i in map(foo, b):
    print(i)
```

```
25
36
49
64
```

[146]:
```
f = map(foo)

for i in f(a):
    print(i)

for i in f(b):
    print(i)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [146], in <cell line: 1>()
----> 1 f = map(foo)
      3 for i in f(a):
      4     print(i)

TypeError: map() must have at least two arguments.
```

[147]:
```
from functools import partial
```

[148]:
```
f = partial(map, foo)
```

[149]:
```
f
```

[149]:
```
functools.partial(<class 'map'>, <function foo at 0x10a8e8ca0>)
```

[150]:
```
for i in f(a):
    print(i)
```

```
1
4
9
16
```

[151]:
```
for i in f(b):
    print(i)
```

```
25
36
49
64
```

[152]:
```
f = partial(map, lambda x: x**3)
```

[153]:
```
for i in f([1, 12, 24]):
    print(i)
```

```
1
1728
13824
```

[154]:
```
def bar(x, y, z):
    return x * y * z
```

[156]:
```
f = partial(bar, 2, 4)
```

[157]:
```
f(3)
```

[157]: 24

[158]:
```
f(4)
```

[158]: 32

[159]:
```
f = partial(bar, 12)
```

[160]:
```
f(2, 2)
```

[160]: 48

[161]:
```
f = lambda k: bar(12, k, 24)
```

[162]:
```
12*24
```

[162]: 288

[163]:
```
f(2)
```

[163]:
```
576
```

[164]:
```
f.__name__
```

[164]:
```
'<lambda>'
```

[167]:
```
def bar(x, y, z):
    """Sample docstring"""
    return x * y * z
```

[168]:
```
g = partial(bar, 12)
```

[169]:
```
g.__doc__
```

[169]:
```
'partial(func, *args, **keywords) - new function with partial application\n
of the given arguments and keywords.\n'
```

[ ]:
```

```