ΠΕΡΙΕΧΟΜΕΝΑ:

1. Decorators Απομνημόνευσης (caching)
2. Πολυμορφισμός: @singledispatch
3. partial και partialmethod
4. @total_ordering και @wraps

ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ:

1. Python Advanced: Μάθημα 4 - Decorators (συναρτήσεις)
2. Python Advanced: Μάθημα 5 - Decorators και Κλάσεις
3. Python Advanced: Μάθημα 6 - Λάμδα και Συν/κός Προγ/μός

Κατερίνα Τ.

Σμαραγδένιος Χορηγός Μαθήματος

'Αλαιν Πατρίκιος

Χρυσός Χορηγός Μαθήματος

ΜΑΘΗΜΑ 6.2: functools

1. Decorators Απομνημόνευσης (Caching)

modules psounis

- Το **module functools** περιέχει χρήσιμους decorators και συναρτήσεις.
- Χρησιμοποιούνται για caching αποτελεσμάτων συναρτήσεων, εύκολο ορισμό σχεσιακών τελεστών σε κλάση κ.α.

- Ο decorator **@lru_cache** διατηρεί στη μνήμη τα πιο πρόσφατα αποτελέσματα της συνάρτησης στην οποία εφαρμόζεται
- Δέχεται ως όρισμα **maxsize** το μέγιστο πλήθος αποτελεσμάτων που θα διατηρήσει (Αν = None, τότε δεν έχει πρακτικό περιορισμό, προκαθορισμένη τιμη = 128)
- Επίσης δέχεται προαιρετικό όρισμα **typed=False**, όπου κάνει διαχωρισμό ανάλογα με τον τύπο δεδομένων (π.χ 42 και 42.0)

Παράδειγμα 1: caching.py

```
from functools import lru_cache

@lru_cache(maxsize=100)
def fib(n):
    if n==0 or n==1:
        return 1
    else:
        return fib(n-1)+fib(n-2)

print(fib(100))
print(fib.cache_info()) # cache stats
fib.cache_clear() # clears cache
```

@cache

- Ισοδύναμος με τον @lru_cache(maxsize=None)

Παράδειγμα 2: cache2.py

```
@cache
def fib(n):
    if n==0 or n==1:
    ...
```

@cached_property

- Είναι ένας συνδυασμός του built-in @property (Python Advanced, Μάθημα 5) και του @cache
- αλλά μόνο για immutable πεδία (των οποίων ο υπολογισμός είναι χρονοβόρος).

Παράδειγμα 3: cache3.py

```
from functools import cached_property
class Calcs:
    def __init__(self):
        self._val = 0
    @cached_property
    def val(self):
        print("some tedious calc...")
        self._val = 100
        return self._val
```

```
calcs = Calcs()
print(calcs.val)
print(calcs.val)
```

ΜΑΘΗΜΑ 6.2: functools

2. Πολυμορφισμός: @singledispatch

- Ο decorator **@singledispatch** χρησιμοποιείται για να μοντελοποιήσει πολυμορφισμό στα ορίσματα μιας συνάρτησης
- Μπαίνει μπροστά από μία συνάρτηση.
- Έπειτα σε διαδοχικές εκδοχές της ίδιας συνάρτησης, καθορίζουμε την συμπεριφορά της συνάρτησης, με διαφορετικούς τύπους δεδομένων.

Παράδειγμα 3: polymorphic_function.py

```
from functools import singledispatch

@singledispatch
def my_print(x):
    raise NotImplementedError("doesn't support this")

@my_print.register
def my_print_var(arg: int):
    print("int")

@my_print.register
def my_print_var(arg: float):
    print("float")

my_print(54)
my_print(2.1)
my_print("str")
```

Σημειώσεις:

- Το όνομα των εκδοχών της συνάρτησης μας δεν έχει σημασία, αρκεί να προσδιορίζεται με το decorator:
  - **@όνομα_συναρτήσης.register**
- Έπειτα ο τύπος δεδομένων προσδιορίζεται ως όρισμα σε κάθε εκδοχή της συνάρτησης

Εντελώς αντίστοιχα ορίζεται ο **@singledispatchmethod** για υπερφόρτωση μεθόδων κλάσης

Παράδειγμα 4: polymorphic_method.py

```
from functools import singledispatchmethod

class MyClass:
    @singledispatchmethod
    def my_print(self, x):
        raise NotImplementedError("doesn't support this")

    @my_print.register
    def my_print_var(self, arg: int):
        print("int")

    @my_print.register
    def my_print_var(self, arg: float):
        print("float")
c = MyClass()
c.my_print(1)
c.my_print(2.1)
c.my_print("str")
```

- Η **συνάρτηση partial** δημιουργεί νέες συναρτήσεις, ορίζοντας προκαθορισμένες τιμές σε ορίσματα ήδη υφιστάμενων συναρτήσεων.
- Σε μία συνάρτηση έχουμε γενικά:
  - Θεσιακά ορίσματα
  - Ορίσματα με λέξεις κλειδιά
- Κατασκευάζουμε τη νέα συνάρτηση ως:
  - **new_func = partial(func, positional_values, keyword_values)**
- και έπειτα την καλούμε θέτοντας τιμές στα ορίσματα που δεν έχουν καθοριστεί στην partial:
  - new_func(arguments)

Παράδειγμα 5: partial.py

```
from functools import partial

def power_func(x, y, a=1, b=0):
    return a*x**y + b

new_func = partial(power_func, 2, a=4)

print(new_func(4, b=1))
print(new_func(1))
```

- Η **συνάρτηση partialmethod** κάνει ακριβώς το ίδιο με την partial.
- Αλλά χρησιμοποιείται για να ορίσει μεθόδους κλάσεων.

Παράδειγμα 6: partial_method.py

```
from functools import partialmethod

class Time:
    def __init__(self, hour, minute, second):
        self.hour = hour
        self.minute = minute
        self.second = second

    def set_hour(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    set_midnight = partialmethod(set_hour, 0, 0, 0)
    set_half = partialmethod(set_hour, minute=30, second=0)


t = Time(11, 1, 2)
print(t)
t.set_midnight()
print(t)
t.set_half(22)
print(t)
```

ΜΑΘΗΜΑ 6.2: functools

4. @total_ordering και @wraps

- Στο μάθημα 17 είδαμε πως να υπερφορτώνουμε σχεσιακούς τελεστές (όπως π.χ. τα ==, >) με τις αντίστοιχες dunder methods (αντ. __eq(..)__, __gt(..)__)
- Ο decorator **@total_ordering** διευκολύνει περαιτέρω αυτή τη διαδικασία:
  - Διακοσμούμε μια κλάση και ορίζουμε σε αυτήν το == και έναν οποιοδήποτε από τους ανισοτικούς τελεστές (π.χ. >)
  - Αυτόματα ορίζονται και όλοι οι υπόλοιποι σχεσιακοί τελεστές.

Παράδειγμα 7: ordering.py

```
from functools import total_ordering
@total_ordering
class Time:
   def __gt__(self, other):
      ...
   def __eq__(self, other):
      ...
t = Time(11,1,2)
t2 = Time(11,1,1)
print(f"{t} > {t2}: {t>t2}")
print(f"{t} < {t2}: {t<t2}")
print(f"{t} >= {t2}: {t>=t2}")
print(f"{t} <= {t2}: {t<=t2}")
print(f"{t} == {t2}: {t==t2}")
print(f"{t} != {t2}: {t!=t2}")
```

- **Πρόβλημα:** Όταν διακοσμούμε μία συνάρτηση, αλλάζει το εσωτερικό όνομά της και το docstring της(!)

- **Λύση: @wraps(function)** στην εσωτερική συνάρτηση (function: αυτή που διοχετεύεται ως όρισμα στο διακοσμητή)

Παράδειγμα 8: wraps.py

```
from functools import wraps

def decorate_with_lines(func):
   @wraps(func)
   def dec():
      """ inner docstring """
      print("=" * 20)
      func()
      print("=" * 20)
   return dec

@decorate_with_lines
def some_func():
   """ some_func docstring """
   print("I did many things.. ")

print(some_func.__name__)
print(some_func.__doc__)
```

Σημείωση:

- Στο “Python Advanced - Μάθημα 6” είδαμε επίσης ότι στο functools περιλαμβάνεται και η σημαντική μέθοδος **reduce**