Coconut: A Novel Language for Functional Programming in Python

by Evan Hubinger

HARVEY MUDD COLLEGE

What Is Coconut?

Coconut is a **variant of Python** built for **simple, elegant, Pythonic functional programming**.

Do I Have to Give Up Python?

Coconut is a strict superset of Python

All valid Python 3 syntax is valid Coconut syntax, which means you can get started writing Coconut just as you would Python.

How Does Coconut Work?

Coconut compiles to **any Python version or implementation**, allowing you to write and maintain one code base for any version of Python you want to run your code on.

Why Use Coconut?

Coconut adds to Python built-in, syntactical support for:

- pattern-matching
- algebraic data types
- destructuring assignment
- partial application
- lazy lists
- function composition
- prettier lambdas
- infix notation
- pipeline-style programming
- operator functions
- tail call optimization
- parallel programming

... and much more!

Coconut Does Pattern-Matching

```
match [head] + tail in [0, 1, 2, 3]:
    print(head, tail)

def factorial(n):
    """Compute n! where n is an integer >= 0."""
    case n:
        match 0:
            return 1
        match _ is int if n > 0:
            return n * factorial(n-1)


def quick_sort([]) = []

@addpattern(quick_sort)
def quick_sort([head] + tail) =
    """Sort a sequence using quick sort."""
    (quick_sort([x for x in tail if x < head])
        + [head]
        + quick_sort([x for x in tail if x >= head]))
```

Coconut Does Algebraic Data Types

```
data vector2(x, y):
    """Immutable two-element vector."""
    def __abs__(self):
        return (self.x**2 + self.y**2)**.5


data Empty()
data Leaf(n)
data Node(l, r)

def size(Empty()) = 0

@addpattern(size)
def size(Leaf(n)) = 1

@addpattern(size)
def size(Node(l, r)) = size(l) + size(r)
```

Coconut Does Tail Call Optimization

```
def factorial(0, acc=1) = acc

@addpattern(factorial)
def factorial(n is int, acc=1 if n > 0) =
    """Compute n! where n is an integer >= 0."""
    factorial(n-1, acc*n)


def is_even(0) = True
@addpattern(is_even)
def is_even(n is int if n > 0) = is_odd(n-1)

def is_odd(0) = False
@addpattern(is_odd)
def is_odd(n is int if n > 0) = is_even(n-1)
```

Coconut Does Much, Much More

```
"hello, world!" |> print

product = reduce$(*)

def a `mod` b = a % b
(x `mod` 2) `print`

{"list": [0] + rest} = {"list": [0, 1, 2, 3]}

def zipwith(f, *args) =
    zip(*args) |> map$(items -> f(*items))

def natural_nums(n=0) =
    """Infinite sequence of natural numbers."""
    (n,) :: natural_nums(n+1)

@recursive_iterator
def fib_seq() =
    """Infinite sequence of fibonacci numbers."""
    (1, 1) :: map((+), fib_seq(), fib_seq()$[1:])

fib_seq()$[:100] |> parallel_map$(pow$(?, 2)) |> list
```