@inline

(or: promise gets pushy)

Ryan Kelly
ryan@rfk.id.au

```
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

```
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

timeit: 52

```
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

```
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += 3*item*item - 2*item + (1 / item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

```
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += 3*item*item - 2*item + (1 / item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

timeit: 36

```
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

```
@inline
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

```
@inline
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

timeit: 40

```
@inline
def calculate(x):
    return 3*x*x - 2*x + (1 / x)


def aggregate(items):
    total = 0
    for item in items:
        total += calculate(item)
    return total

def testit():
    return aggregate(xrange(1,1000))
```

timeit: [218, 40, 40]

How?

byteplay

http://code.google.com/p/byteplay/

don't hack bytecode without it

```
def inline(func):
    c = Code.from_code(func.func_code)

    c.code.insert(0,(LOAD_CONST,_inlineme))
    c.code.insert(1,(LOAD_CONST,func))
    c.code.insert(2,(CALL_FUNCTION,1))
    c.code.insert(3,(POP_TOP,None))

    func.func_code = c.to_code()
    return func
```

```
def _inlineme(func):


    myframe = sys._getframe(1)
    assert _find_function(myframe) == func
```

```
def _inlineme(func):


    myframe = sys._getframe(1)
    assert _find_function(myframe) == func

    callframe = sys._getframe(2)
    caller = _find_function(callframe)
    callsite = callframe.f_lasti
```

```
def _inlineme(func):


    myframe = sys._getframe(1)
    assert _find_function(myframe) == func

    callframe = sys._getframe(2)
    caller = _find_function(callframe)
    callsite = callframe.f_lasti

    old_code = Code.from_code(caller.func_code)
    assert old_code.code[callsite][0] == CALL_FUNCTION
```

```
def _inlineme(func):


    myframe = sys._getframe(1)
    assert _find_function(myframe) == func

    callframe = sys._getframe(2)
    caller = _find_function(callframe)
    callsite = callframe.f_lasti

    old_code = Code.from_code(caller.func_code)
    assert old_code.code[callsite][0] == CALL_FUNCTION

    new_code = _inline_code(old_code,func.func_code,callsite)
    caller.func_code = new_code.to_code()
```

```
>>> dis.dis(aggregate)
```

```
>>> dis.dis(aggregate)
 10           0 LOAD_CONST               1 (0)
              3 STORE_FAST               1 (total)

 11           6 SETUP_LOOP              30 (to 39)
              9 LOAD_FAST                0 (items)
             12 GET_ITER
        >>   13 FOR_ITER                22 (to 38)
             16 STORE_FAST               2 (item)

 12          19 LOAD_FAST                1 (total)
             22 LOAD_GLOBAL              0 (calculate)
             25 LOAD_FAST                2 (item)
             28 CALL_FUNCTION            1
             31 INPLACE_ADD
             32 STORE_FAST               1 (total)
             35 JUMP_ABSOLUTE           13
        >>   38 POP_BLOCK

 14     >>   39 LOAD_FAST                1 (total)
             42 RETURN_VALUE
```

```
>>> dis.dis(aggregate)
              ...
 12           19 LOAD_FAST                1 (total)
              22 LOAD_GLOBAL              0 (calculate)
              25 LOAD_FAST                2 (item)
              28 CALL_FUNCTION            1
              31 INPLACE_ADD
              32 STORE_FAST               1 (total)
              35 JUMP_ABSOLUTE           13
              ...
```

```
>>> aggregate([1,2,3])
31
>>> dis.dis(aggregate)
```

```
>>> aggregate([1,2,3])
31
>>> dis.dis(aggregate)
              ...
              19 LOAD_FAST                1 (total)
              22 LOAD_FAST                2 (item)
              25 STORE_FAST               3 (_inlined_var1_x)
              28 LOAD_CONST               2 (3)
              31 LOAD_FAST                3 (_inlined_var1_x)
              34 BINARY_MULTIPLY
              35 LOAD_FAST                3 (_inlined_var1_x)
              38 BINARY_MULTIPLY
              39 LOAD_CONST               3 (2)
              42 LOAD_FAST                3 (_inlined_var1_x)
              45 BINARY_MULTIPLY
              46 BINARY_SUBTRACT
              47 LOAD_CONST               4 (1)
              50 LOAD_FAST                3 (_inlined_var1_x)
              53 BINARY_DIVIDE
              54 BINARY_ADD
              55 JUMP_ABSOLUTE           58

              ...
```

Disclaimers...

1) You probably don't need to optimise

1) You probably don't need to optimise

2) You probably don't need to optimise *that*

1) You probably don't need to optimise

2) You probably don't need to optimise *that*

3) You probably need a better algorithm

1) You probably don't need to optimise

2) You probably don't need to optimise *that*

3) You probably need a better algorithm

4) Or a JIT

5) Don't ugly up your code

come fork me:

http://github.com/rfk/promise/