Intro to Python (Class 6)

Classes in Python

Ben Bettisworth

2025-06-12

Outline

Intro 1

A Reminder About Computers 2

Example 4

Encapsulation 8

Classes 12

Syntax 13

Class Data 16

Methods 20

Object Oriented Programming 23

What is an Object 24

Object Oriented Programming 26

Private vs Public 35

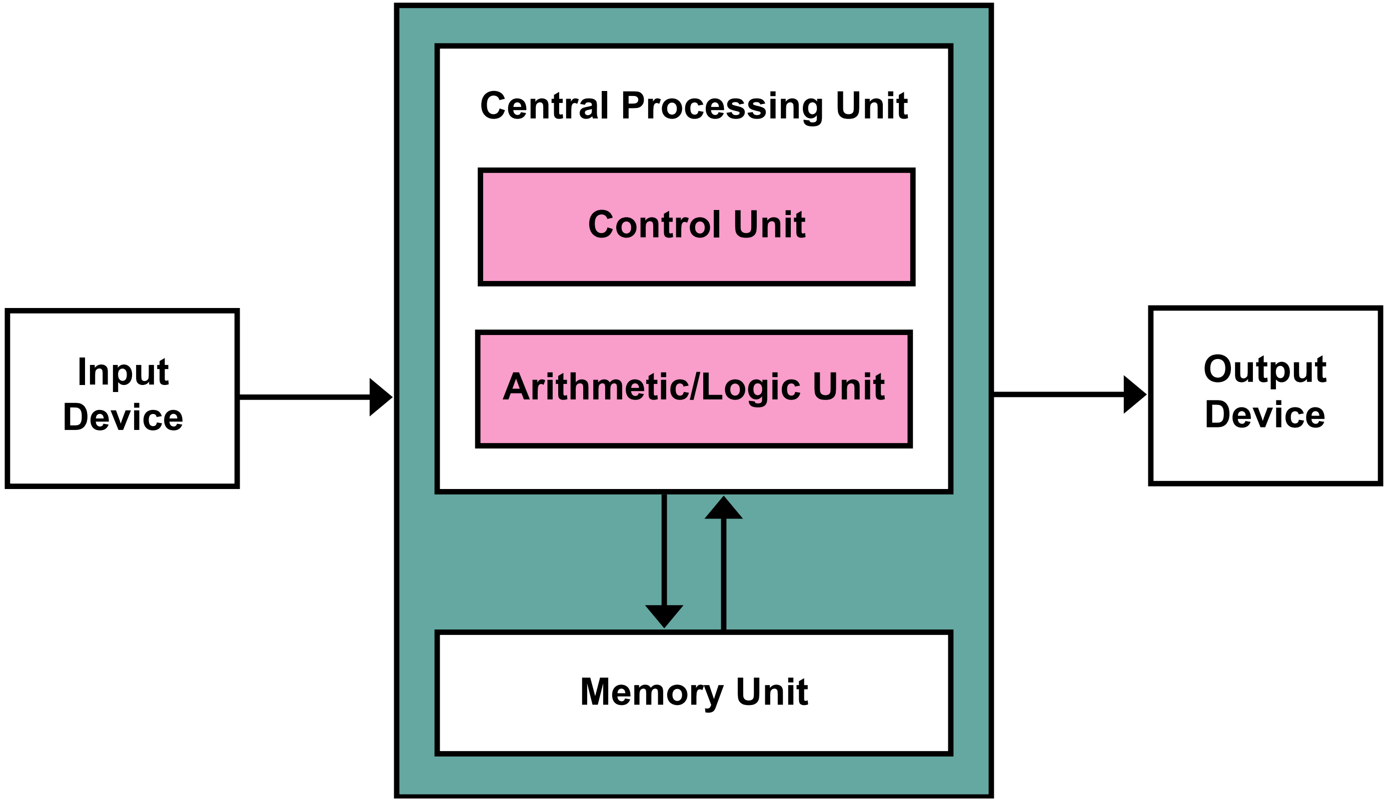
Misc 39

Everything is an Object 40

There are other programming styles 42

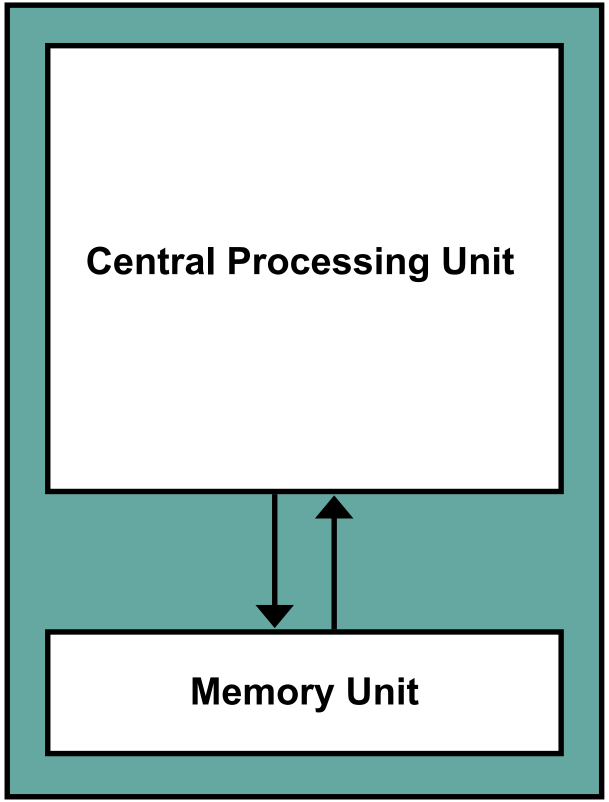
Final thoughts 43

A Reminder About Computers



A Reminder About Computers

The Part that Matters



Example

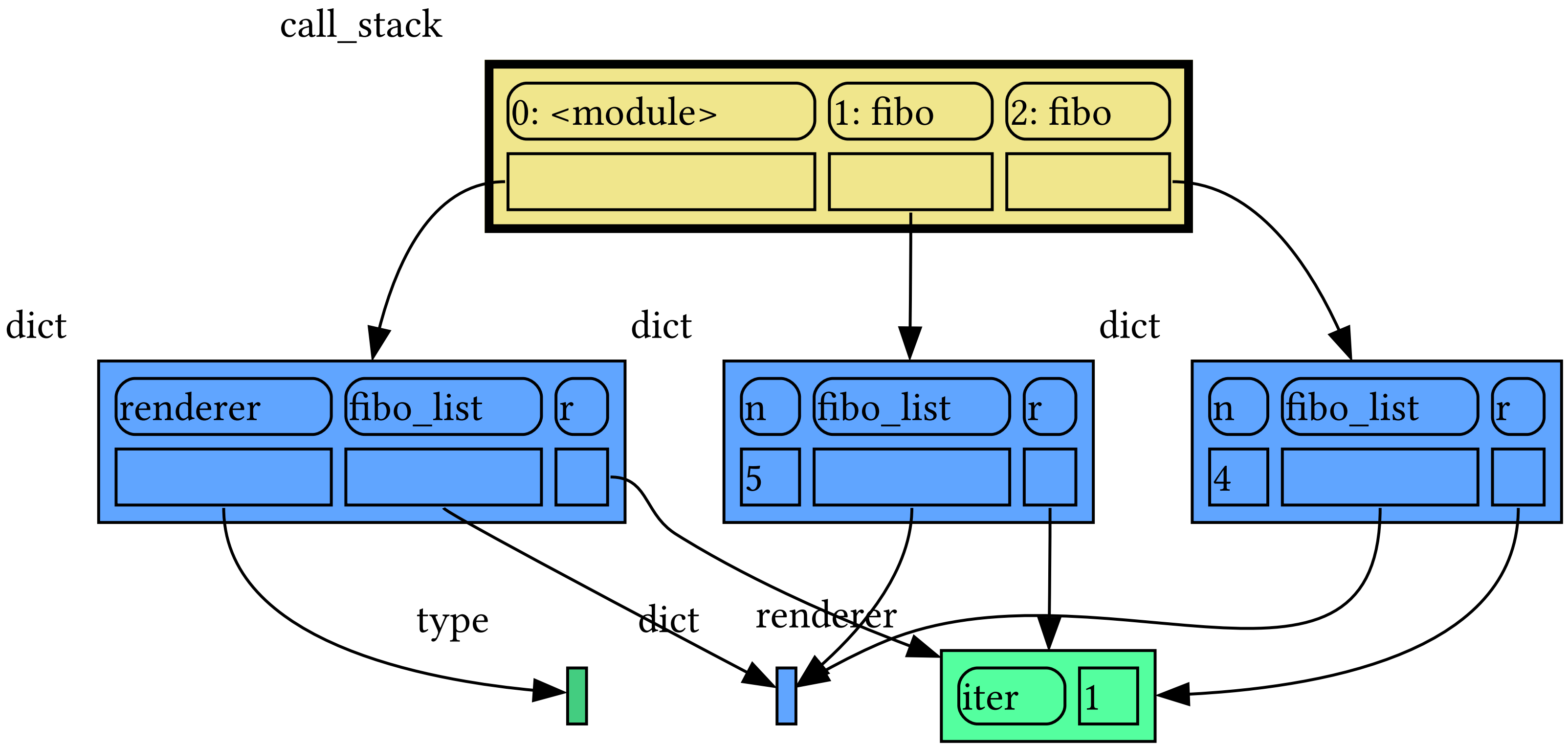
Another Fibo

```python
def fibo(n, fibo_list, r):
    r.save(mg.stack(), "state")

    if n == 0 or n == 1:
        result = 1
    else:
        result = fibo(n-1, fibo_list, r) + fibo(n-2, fibo_list, r)
    fibo_list[n] = result
    return result
```

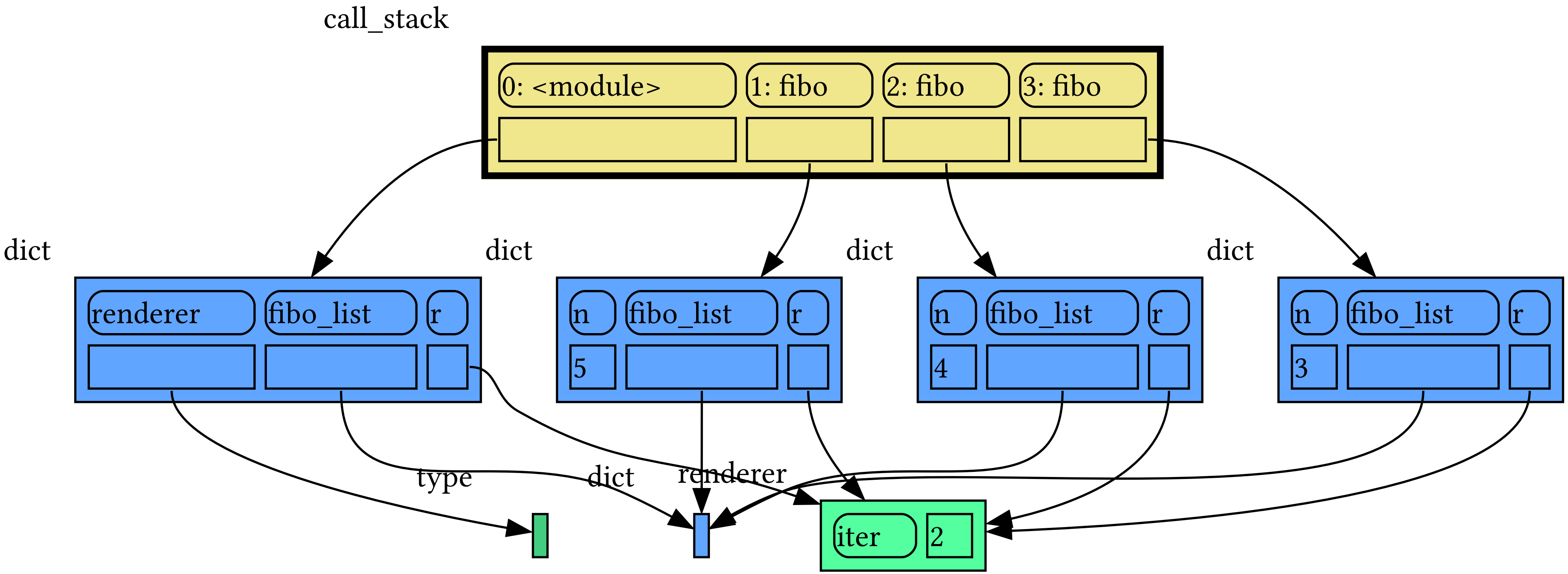
Example

```python
class renderer:
    def __init__(self):
        self.iter = 0

    def save(self, data, prefix):
        mg.render(data, f"{prefix}-{self.iter}.svg")
        self.iter += 1
```

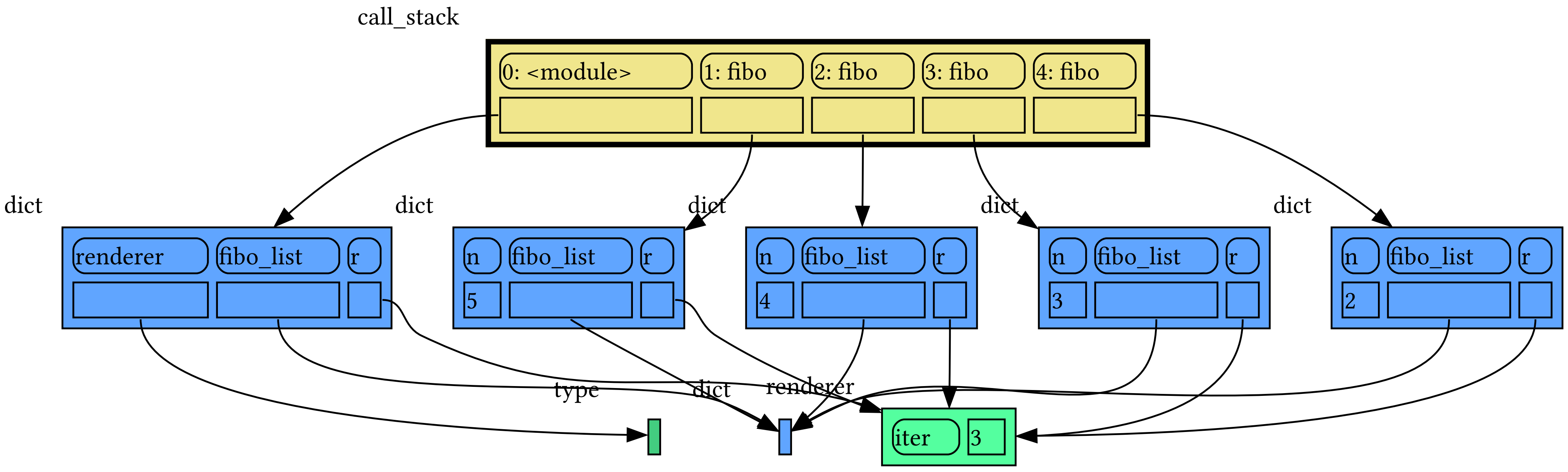
Example


call_stack
0: <module>
1: fibo
2: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
type
dict
renderer
iter
1

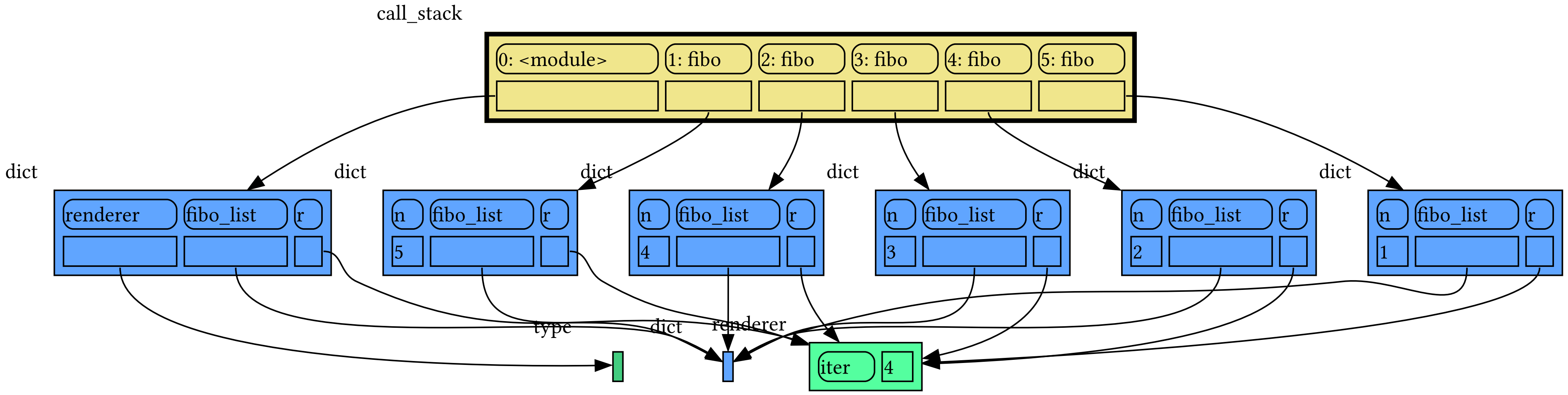
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
3
type
dict
renderer
iter
2

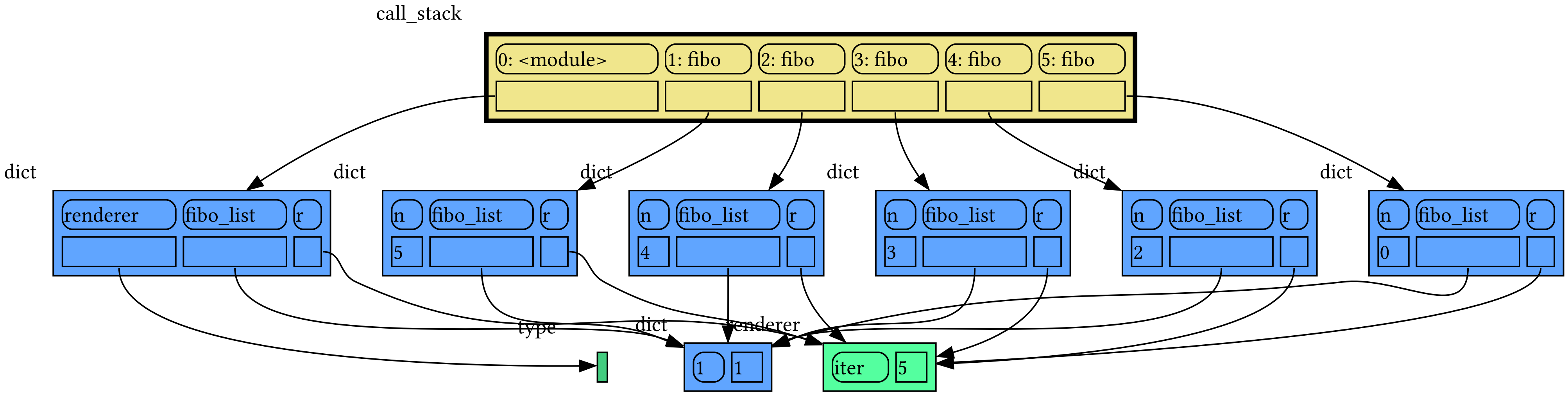
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
3
dict
n
fibo_list
r
2
type
dict
renderer
iter
3

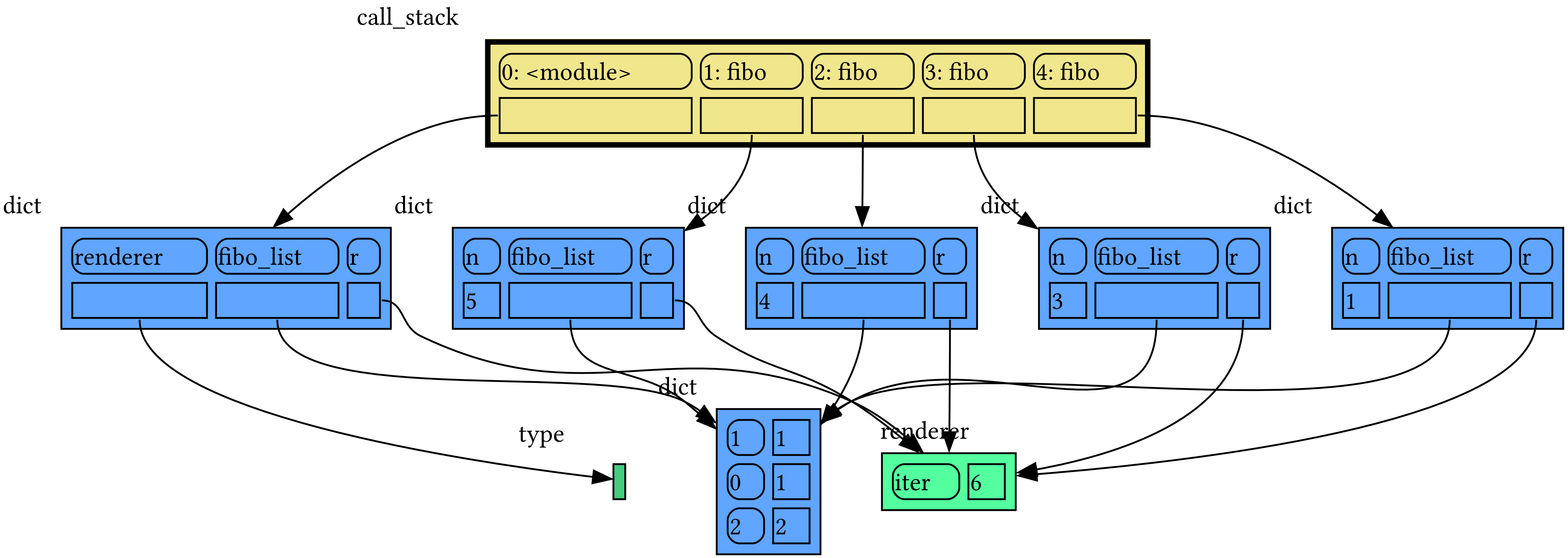
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
5: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
3
dict
n
fibo_list
r
2
dict
n
fibo_list
r
1
type
dict
renderer
iter
4

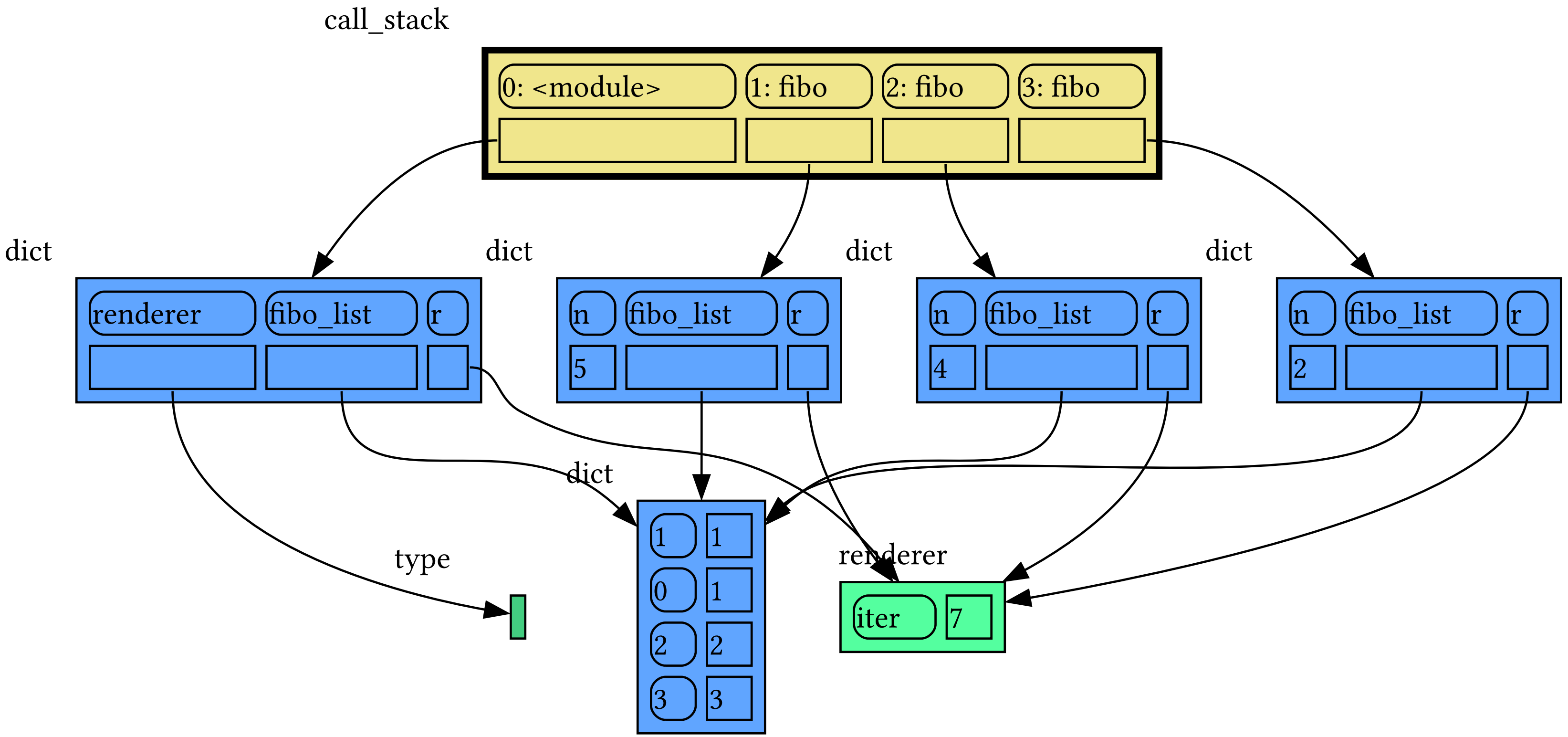
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
5: fibo
dict
dict
dict
dict
dict
dict
renderer
fibo_list
r
n
fibo_list
r
5
n
fibo_list
r
4
n
fibo_list
r
3
n
fibo_list
r
2
n
fibo_list
r
0
type
dict
renderer
1
1
iter
5

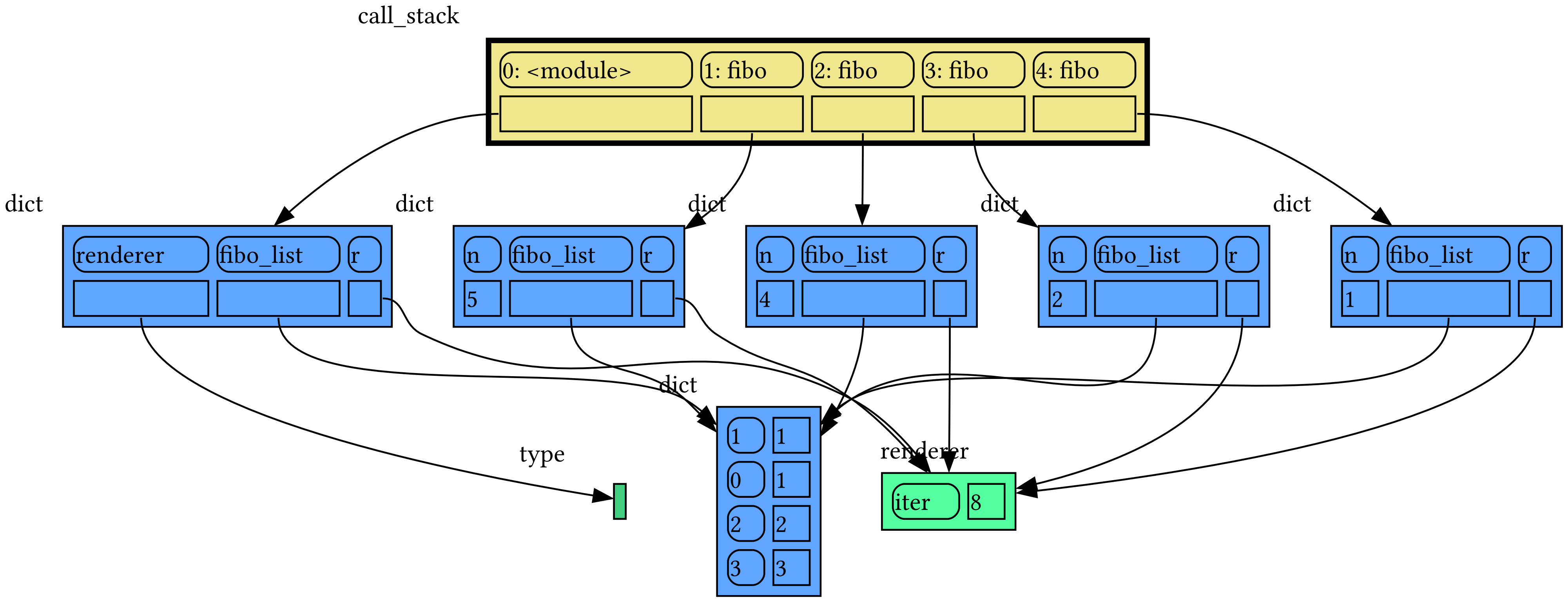
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
3
dict
n
fibo_list
r
1
type
dict
1
1
0
1
2
2
renderer
iter
6

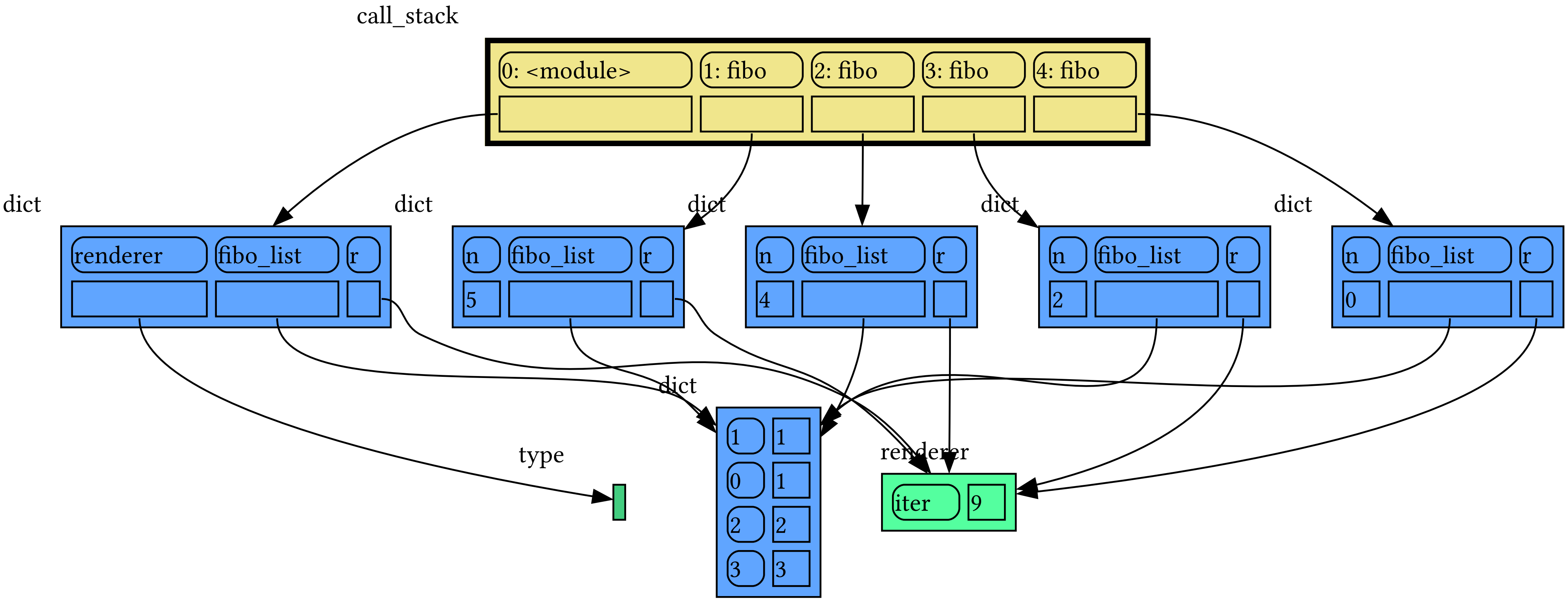
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
2
dict
1
1
0
1
2
2
3
3
type
renderer
iter
7

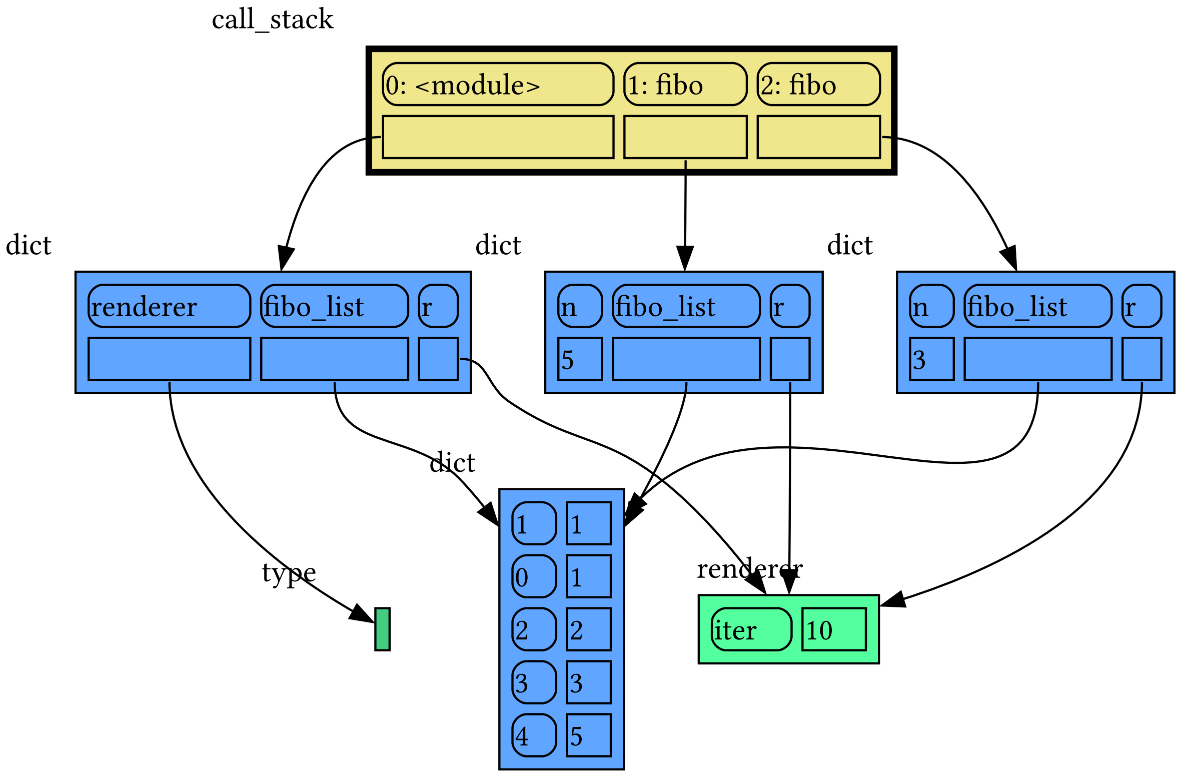
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
2
dict
n
fibo_list
r
1
type
dict
1
1
0
1
2
2
3
3
renderer
iter
8

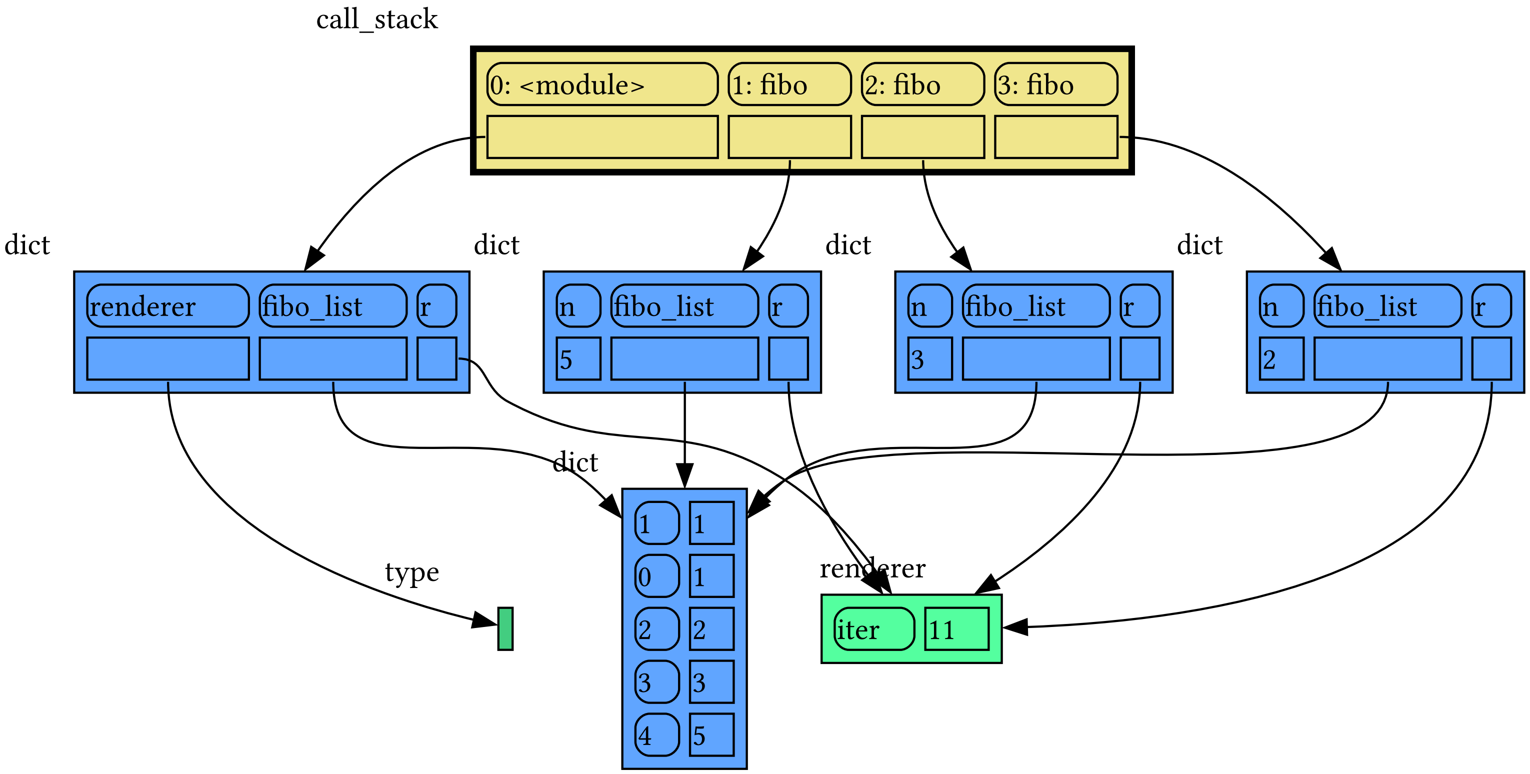
Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
4
dict
n
fibo_list
r
2
dict
n
fibo_list
r
0
dict
1
1
0
1
2
2
3
3
type
renderer
iter
9

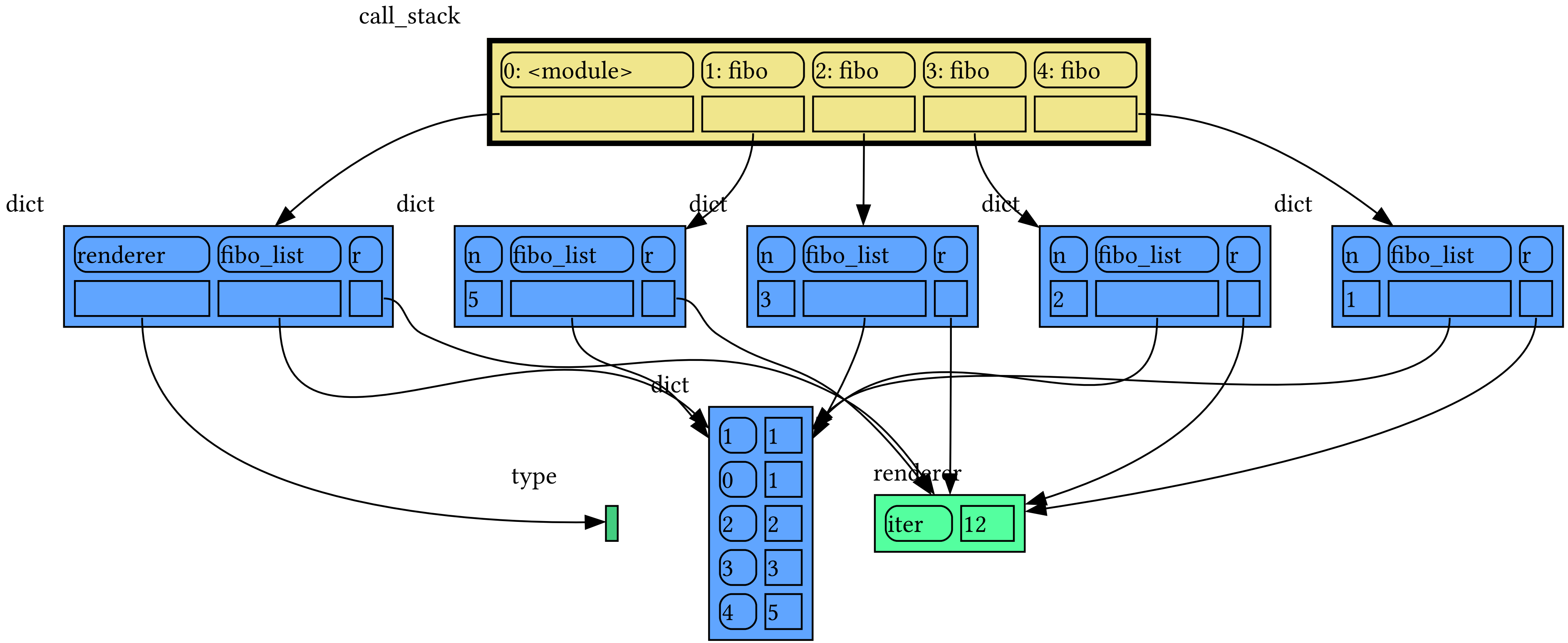
Example


call_stack
0: <module>
1: fibo
2: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
3
dict
1
1
0
1
2
2
3
3
4
5
type
renderer
iter
10

Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
3
dict
n
fibo_list
r
2
dict
1
1
0
1
2
2
3
3
4
5
type
renderer
iter
11

Example


call_stack
0: <module>
1: fibo
2: fibo
3: fibo
4: fibo
dict
renderer
fibo_list
r
dict
n
fibo_list
r
5
dict
n
fibo_list
r
3
dict
n
fibo_list
r
2
dict
n
fibo_list
r
1
dict
1
1
0
1
2
2
3
3
4
5
type
renderer
iter
12

Example

Conclusion

State Definition 1

- State is hard
- State is your worst nightmare
- All you do is manage state

Encapsulation

A Solution

- A good way to handle this problem is to break up the problem into smaller parts.
- Each of these parts manages some part of the state.

Encapsulation

A Solution

- A good way to handle this problem is to break up the problem into smaller parts.
- Each of these parts manages some part of the state.

Encapsulation Definition 2

We say state is encapsulated when it is managed by some module of the program

Encapsulation

Examples

You have already encountered some examples of encapsulation:

- `list`s
- `dict`s
- Functions

Encapsulation

The Tool

We have the final tool in encapsulation:

`class` Definition 3

In Python, a `class` is an object that contains data, and functions on that data.

```python
class foo:
  def __init__(self):
    self._bar = "howdy"
  def change_bar(self, new_bar):
    self._bar = new_greeting
```

Encapsulation

```python
class foo:
    def __init__(self):
        self._bar = "howdy"
    def change_bar(self, new_bar):
        self._bar = new_greeting
    def greet(self):
        print(self._bar)
```

(1) Setup State

(2) Modify State

(3) Use State

Outline

Intro ... 1

A Reminder About Computers ... 2

Example ... 4

Encapsulation ... 8

Classes ... 12

Syntax ... 13

Class Data ... 16

Methods ... 20

Object Oriented Programming ... 23

What is an Object ... 24

Object Oriented Programming ... 26

Private vs Public ... 35

Misc ... 39

Everything is an Object ... 40

There are other programming styles ... 42

Final thoughts ... 43

Syntax

```python
class Counter:
  """A class to count things"""

  def __init__(self):
    self._counter = 0

  def count(foo):
    for i in foo:
      self._counter += 1
```

(1) Class Declaration

(2) Class Initialization

(3) Methods

Syntax

```python
class Counter:
    """A class to count things"""

    def __init__(self):
        self._counter = 0

    def count(self, foo):
        for i in foo:
            self._counter += 1
```

} (1) Instance Variable

} (2) Managing State

Syntax

```
c = Counter()
c.count([1,2,3,4])
print(c._counter)

c.count([i for i in range(10)])
print(c._counter)
```

Syntax

```python
c = Counter()
c.count([1,2,3,4])
print(c._counter)

c.count([i for i in range(10)])
print(c._counter)
```

Code Result

Output 5

```
4
14
```

Class Data

Instance

Definition 6

An instance is a particular bound class.

```
c1 = Counter()
c2 = Counter()
```

Instance Variable

Definition 7

An instance variable are variables that are bound to an instance of a class.

```
id(c1._counter) == id(c2._counter)
>>> False
```

Class Data

`self` is a reference to the current class. It is passed as the first argument of methods.

Using a Method

Example 8

```python
class Counter:
  """A class to count things"""
…
  def count(self, foo):
    for i in foo:
      self._counter += 1
```

Here, `self` is bound to and instance of `Counter`. For example `c1`.

Class Data

`self` is a reference to the current class. It is passed as the first argument of methods.

Using a Method Example 9

```python
class Counter:
  """A class to count things"""
…
  def count(self, foo):
    for i in foo:
      self._counter += 1
```

Using a Function Example 10

```python
def count(c, foo):
  for i in foo:
    c1._count += 1

c1 = Counter()
count(c1, [1, 2])
```

Example 9 and 10 are functionally equivalent.

Class Data

In addition to instance variables, there are class variables.

Class Variable Definition 11

A class variable is a variable which is bound to all instances of the same class

WideCounter Example 12

```python
class WideList:
  _list = []
  def __init__(self):
    pass
```

Class Data

Using `WideCounter` — Example 13

```python
wc1 = WideCounter()
wc2 = WideCounter()
wc1._list.append(1)
print(f"wc1: {wc1._counter}, wc2: {wc2._counter}")
```

Demonstrating Class Variables — Output 14

```
wc1: [1], wc2: [1]
```

Methods

Method — Definition 15

A method is a function which operates in the context of a class. In Python, a method will always have `self` as the first argument.

Method — Example 16

```
class Counter:
…                                         …
  def count(self, foo):
    for i in foo:
      self._counter += 1
```

Methods

Method Definition 17

A method is a function which operates in the context of a class. In Python, a method will always have `self` as the first argument.

The scope of a method is just like any other function. In particular, other class methods and variables must be accessed via `self`.

Methods

Methods are accessed like functions, but they use a `.` operator.

Using Methods Example 18

```python
c1 = Counter()
c1.count([1,2,3])
```

Methods

Methods are accessed like functions, but they use a `.` operator.

Using Methods Example 19

```
1  c1 = Counter()
2  c1.count([1,2,3])
```

Line 2 calls `count` with the arguments `c1` and `[1,2,3]`.

Methods

Magic Methods Definition 20

Magic methods (also known as dunder methods) are special methods that are blessed by python with special syntax.

Methods

Magic Methods

Definition 21

Magic methods (also known as dunder methods) are special methods that are blessed by python with special syntax.

Magic Methods 1

Example 22

```
class Counter:
…                                              …
  def __init__(self):
    self._counter = 0
```

Methods

Magic Methods — Definition 23

Magic methods (also known as dunder methods) are special methods that are blessed by python with special syntax.

Magic Methods 2 — Example 24

```python
class Counter:
  def __add__(self, other):
    c = Counter()
    c._counter = self._counter + other._counter
    return c
```

Methods

Magic Methods

Definition 25

Magic methods (also known as dunder methods) are special methods that are blessed by python with special syntax.

Magic Methods 2

Example 26

```python
c1 = Counter()
c2 = Counter()
c3 = c1 + c2
```

Outline

Intro 1

- A Reminder About Computers 2
- Example 4
- Encapsulation 8

Classes 12

- Syntax 13
- Class Data 16
- Methods 20

Object Oriented Programming 23

- What is an Object 24
- Object Oriented Programming 26
- Private vs Public 35

Misc 39

- Everything is an Object 40
- There are other programming styles 42
- Final thoughts 43

What is an Object

When using classes, it is helpful to use Object Oriented Programming.

Object Definition 27

An object is the representation of a concept. In practice, they are implemented with `class`.

Example Object Example 28

```python
class Box:
  def __init__(self, w, h):
    self._width, self._height = (w, h)
…
```

What is an Object

When using classes, it is helpful to use Object Oriented Programming.

Object Definition 29

An object is the representation of a concept. In practice, they are implemented with `class`.

Example Object Example 30

```python
class Box:
…
  def area(self):
    return self._width * self._height
```

What is an Object

Since we have labeled the parts of `Box`, we no longer have to make sure we keep what each number means in our head.

Code Organization — Tip 31

By organizing your code into objects, you make the intention of your code more clear. This makes it easier to find bugs.

Object Oriented Programming

Object Oriented Programming Definition 32

Object Oriented Programming is when a program is organized into objects which interact with each other through channels.

Object Oriented Programming

Object Oriented Programming Definition 33

Object Oriented Programming is when a program is organized into objects ~~which interact with each other through channels~~.

Object Oriented Programming

Object Oriented Programming Definition 34

Object Oriented Programming is when ~~a program is organized into~~ objects ~~which interact with each other through~~ ~~channels~~.

Object Oriented Programming

Object Oriented Programming Definition 35

Object Oriented Programming is when objects.

Object Oriented Programming

Objects as Concepts

The central idea in OOP is to divide the “world” into objects, and to define the tasks as operations on those objects.

Object Oriented Programming

Objects as Concepts

The central idea in OOP is to divide the “world” into objects, and to define the tasks as operations on those objects.

In practice, this means defining `class`s that manage data, and implementing methods which do the tasks you want.

Object Oriented Programming

Objects as Concepts

The central idea in OOP is to divide the “world” into objects, and to define the tasks as operations on those objects.

In practice, this means defining `class`s that manage data, and implementing methods which do the tasks you want.

Your program then is a composition of these objects and methods on these objects.

Object Oriented Programming

Applied OOP — Example 36

Suppose you are tasked with writing a program to classify reads. Your boss says you must do this by mapping the reads into a reference database. How should break the program down into objects/`class`?

Object Oriented Programming

- A `reference` class, which has:
  - a `label`, indicating the taxa, and
  - a `sequence`, containing the DNA sequence.
- A `read` class, which has:
  - a `source`, indicating the source,
  - an `assignment` , a `reference` which the read is assigned to with a score, and
  - a `sequence`, containing the DNA sequence.

Object Oriented Programming

- A `reference` class, which has:
  - a `label`, indicating the taxa, and
  - a `sequence`, containing the DNA sequence.
- A `read` class, which has:
  - a `source`, indicating the source,
  - an `assignment` , a `reference` which the read is assigned to with a score, and
  - a `sequence`, containing the DNA sequence.

- `reference` has a method `score` which takes a `read`, and returns a score for that `reference` `read` pair.
- `read` has a method `assign` which takes a `reference` and a score, and assigns the `read` if the new `reference` is better.

Object Oriented Programming

Objects as Managers of State

In the previous example, we managed the state of read assignment by letting each `read` track which `reference` they are assigned to.

Object Oriented Programming

Objects as Managers of State

In the previous example, we managed the state of read assignment by letting each `read` track which `reference` they are assigned to.

This might be preferable to a method that uses multiple arrays, as you don't have to keep track of the indexes.

Object Oriented Programming

Objects as Managers of State

In the previous example, we managed the state of read assignment by letting each `read` track which `reference` they are assigned to.

This might be preferable to a method that uses multiple arrays, as you don't have to keep track of the indexes.

In this way, you interact with a single `read` at a time which keeps track of its own assignment.

Object Oriented Programming

Inheritance

You might have noticed that in the previous example, both `reference` and `read` had `sequence` instance variables.

Object Oriented Programming

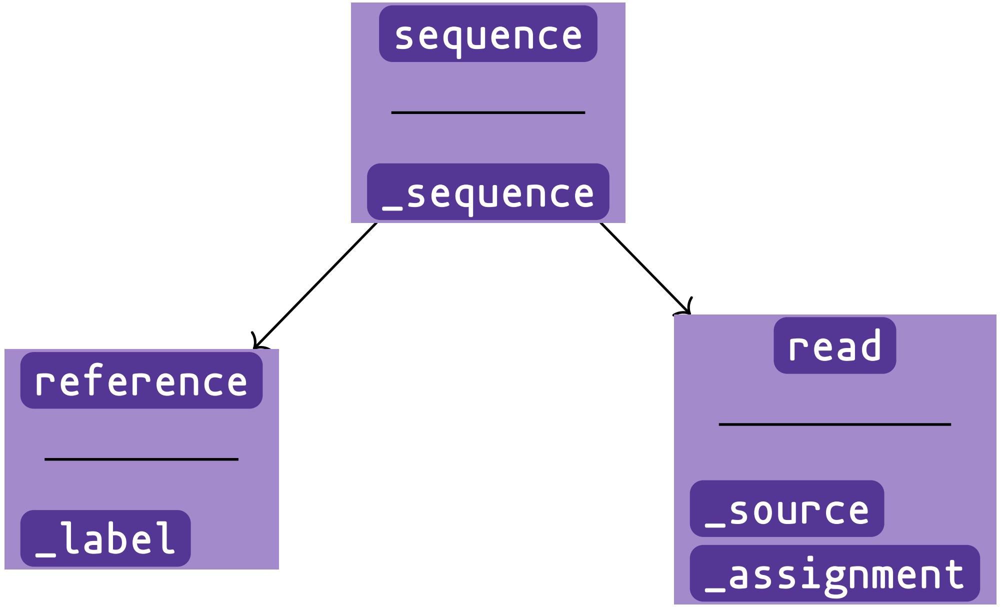
Inheritance

You might have noticed that in the previous example, both `reference` and `read` had `sequence` instance variables.

```python
class sequence:
  def __init__(self, seq):
    self._sequence = seq
  def distance(self, other):
    score = 0
    for i, j in zip(self._sequence, other._sequence):
      if i != j:
        score += 1
    return score
```

Object Oriented Programming

Inheritance



Object Oriented Programming

Inheritance in Practice

Example 37

```python
class read(sequence):
  def __init__(self, source, sequence):
    self._assignment = None
    self._source = source
    super().__init__(sequence)

class reference(sequence):
  def __init__(self, label, sequence):
    self._label = label
    super().__init__(sequence)
```

Object Oriented Programming

Inheritance in Practice 2

Example 38

Suppose we have a `read` `r` and a `reference` `ref`. Then we can write the following

```
score = ref.distance(r)
print(score)
>>> 325
```

Please note that we never defined `distance` for the `reference` class. It was inherited from `sequence`.

Private vs Public

In order to maintain state, we make a distinction between public and private methods and variables.

Private vs Public

In order to maintain state, we make a distinction between public and private methods and variables.

Public and Private Definition 40

A public member is something that is accessible from outside the class.

A private member is something that is accessible only from inside the class.

Private vs Public

In order to maintain state, we make a distinction between public and private methods and variables.

Public and Private Definition 41

A public member is something that is accessible from outside the class.

A private member is something that is accessible only from inside the class.

Python does not have a true distinction between public and private. Instead, members are given `_` as a leading character to mark them as private.

Private vs Public

Interacting with private members

To interact with private members, we tend to write getters and setters

Getters and Setters

Definition 42

A getter method is a method which gets the value (or some computed value) from a private member.

A setter method is a method which sets the value (or some computed value) to a private member.

Private vs Public

Getters and Setters

Example 43

```python
class read:
  def assign(self, reference, score):
    if self._score < score:
      self._reference = reference
      self._score = score
```

Here, `assign` is a setter. It is used to set the private variables `_reference` and `_score`.

Private vs Public

Static Methods

Static methods are to methods as class variables are to instance variables.

Static Method Definition 44

A static method is a method which does not take an instance as it’s first argument.

Private vs Public

Static Methods

Static methods are to methods as class variables are to instance variables.

Static Method — Example 45

```python
class sequence:
  @staticmethod
  def alphabet():
    return ['A', 'C', 'T', 'G']
```

Outline

Everything is an Object

In python, (nearly) everything is an object!

Some Wacky Stuff Example 46

```
def foo():
  print(foo.bar)
foo.bar = "howdy yall"
foo()
>>> "howdy yall"
```

@dataclass

Example 47

```python
@dataclass
class read:
    type: str
    sequence: str
    source: str
    assignment: reference
    score: float
```

Here, python will do most of the boilerplate for you, implementing an `__init__` class for you, and turning the class variables into instance variables.

There are other programming styles

While OOP is popular still, it sometimes is not the best solution. Consider learning about some of the following styles:

- Functional Programming
- Imperative Programming
- Data Driven Design

Final thoughts

I don’t have any

Any questions???????!?!?!??