Introduction to Python

DataLab

November 5, 2016

Outline

Outline

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

- "Things should be as simple as possible, but no simpler" (Einstein)
- Do one thing well (The "UNIX philosophy")
- Don't fret too much about performance–plan to optimize later when needed
- Don't fight the environment and go with the flow
- Don't try for perfection because "good enough" is often just that

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

- "Things should be as simple as possible, but no simpler" (Einstein)
- Do one thing well (The "UNIX philosophy")
- Don't fret too much about performance– plan to optimize later when needed
- Don't fight the environment and go with the flow
- Don't try for perfection because "good enough" is often just that

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

Philosophy

- Borrow ideas from elsewhere whenever it makes sense.
- "Things should be as simple as possible, but no simpler." (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don't fret too much about performance–plan to optimize later when needed.
- Don't fight the environment and go with the flow.
- Don't try for perfection because "good enough" is often just that.

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

Philosophy

- Borrow ideas from elsewhere whenever it makes sense.
- "Things should be as simple as possible, but no simpler." (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don't fret too much about performance--plan to optimize later when needed.
- Don't fight the environment and go with the flow.
- Don't try for perfection because "good enough" is often just that.

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

Philosophy

- Borrow ideas from elsewhere whenever it makes sense.
- "Things should be as simple as possible, but no simpler." (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don't fret too much about performance–plan to optimize later when needed.
- Don't fight the environment and go with the flow.
- Don't try for perfection because "good enough" is often just that.

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

Philosophy

- Borrow ideas from elsewhere whenever it makes sense.
- "Things should be as simple as possible, but no simpler." (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don't fret too much about performance--plan to optimize later when needed.
- Don't fight the environment and go with the flow.
- Don't try for perfection because "good enough" is often just that.

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

Philosophy

- Borrow ideas from elsewhere whenever it makes sense.
- "Things should be as simple as possible, but no simpler." (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don't fret too much about performance--plan to optimize later when needed.
- Don't fight the environment and go with the flow.
- Don't try for perfection because "good enough" is often just that.

Python has a Long Story

Conceived by Guido Van Rossum in the late 1980's

Benevolent dictator for life!!!

From 1989 to 1994

Many of the mayor features of the Python Language were put in place.

Philosophy

- Borrow ideas from elsewhere whenever it makes sense.
- "Things should be as simple as possible, but no simpler." (Einstein)
- Do one thing well (The "UNIX philosophy").
- Don't fret too much about performance--plan to optimize later when needed.
- Don't fight the environment and go with the flow.
- Don't try for perfection because "good enough" is often just that.

Outline

In 1994 (Python 1.0) Functional Tools were Included

$\lambda-$Calculus

The formal system in mathematical logic for expressing computation based on

1. Function Abstraction.
2. Application using binding and substitution.

In 1994 (Python 1.0) Functional Tools were Included

$\lambda-$Calculus

The formal system in mathematical logic for expressing computation based on

1. Function Abstraction.
2. Application using binding and substitution.

Mapping

It applies an specific function to every item of iterable and return a list of the results.

In 1994 (Python 1.0) Functional Tools were Included

$\lambda-$Calculus

The formal system in mathematical logic for expressing computation based on

1. Function Abstraction.
2. Application using binding and substitution.

Mapping

It applies an specific function to every item of iterable and return a list of the results.

Filters

It filters out items based on a test function.

In 1994 (Python 1.0) Functional Tools were Included

$\lambda-$Calculus

The formal system in mathematical logic for expressing computation based on

1. Function Abstraction.
2. Application using binding and substitution.

Mapping

It applies an specific function to every item of iterable and return a list of the results.

Filters

It filters out items based on a test function.

In 1994 (Python 1.0) Functional Tools were Included

$\lambda-$Calculus

The formal system in mathematical logic for expressing computation based on

1. Function Abstraction.
2. Application using binding and substitution.

Mapping

It applies an specific function to every item of iterable and return a list of the results.

Filters

It filters out items based on a test function.

Furthermore

Reduce

A function for performing some computation on a list and returning the result.

Outline

We have several

CPython

- This is the classic and we will reference it every time we explain something!!!

- Quite interesting to use a semi-interpreted language on top of a virtual machine

- PyPy (2007) written in RPython

We have several

CPython

- This is the classic and we will reference it every time we explain something!!!

Jython

- A Java implementation of the Python.
- Quite interesting to use a semi-interpreted language on top of a virtual machine.

- PyPy (2007) written in RPython

We have several

CPython

- This is the classic and we will reference it every time we explain something!!!

Jython

- A Java implementation of the Python.
- Quite interesting to use a semi-interpreted language on top of a virtual machine.

There are others

- IronPython (2006) - .NET
- PyPy (2007) written in RPython

We have several

CPython

- This is the classic and we will reference it every time we explain something!!!

Jython

- A Java implementation of the Python.
- Quite interesting to use a semi-interpreted language on top of a virtual machine.

There are others

- IronPython (2006) - .NET
- PyPy (2007) written in RPython

We have several

CPython

- This is the classic and we will reference it every time we explain something!!!

Jython

- A Java implementation of the Python.
- Quite interesting to use a semi-interpreted language on top of a virtual machine.

There are others

- IronPython (2006) - .NET
- PyPy (2007) written in RPython

Outline

Is Python Interpreted or Compiled?

Python is compiled

- Not compiled to machine code ahead of time
  - "only" compiled to bytecode!!!

- Or as in PyPy, both interpreted and compiled to optimized machine code at runtime

Is Python Interpreted or Compiled?

Python is compiled

- Not compiled to machine code ahead of time
  - "only" compiled to bytecode!!!

That bytecode is either interpreted

- As with the reference implementation (CPython).
- Or as in PyPy, both interpreted and compiled to optimized machine code at runtime.

Is Python Interpreted or Compiled?

Python is compiled

- Not compiled to machine code ahead of time
  - "only" compiled to bytecode!!!

That bytecode is either interpreted

- As with the reference implementation (CPython).
- Or as in PyPy, both interpreted and compiled to optimized machine code at runtime.

Is Python Interpreted or Compiled?

Python is compiled

- Not compiled to machine code ahead of time
  - "only" compiled to bytecode!!!

That bytecode is either interpreted

- As with the reference implementation (CPython).
- Or as in PyPy, both interpreted and compiled to optimized machine code at runtime.

Outline

The Memory Management System

Why do we care about this?

- Once you know the memory management system
    - You can build more efficient code
    - How does this fact affect Python?

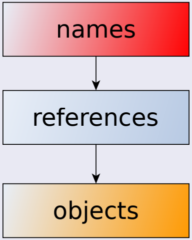
You have NAMES pointing to references and objects

The Memory Management System

Why do we care about this?

- Once you know the memory management system
  - You can build more efficient code
  - How does this fact affect Python?

You have NAMES pointing to references and objects



What do you have?

Memory management in Python

- It involves a **private heap** containing all Python objects and data structures.
- The management of this private heap is ensured internally by the Python memory manager.

What do you have?

Memory management in Python

- It involves a **private heap** containing all Python objects and data structures.
- The management of this private heap is ensured internally by the Python memory manager.

Thus, we have that the Python Interpreter

- Uses what is called a reference count.

What do you have?

Memory management in Python

- It involves a **private heap** containing all Python objects and data structures.
- The management of this private heap is ensured internally by the Python memory manager.

Thus, we have that the Python Interpreter

- Uses what is called a reference count.

A Reference Count?

"**In computer science, reference counting is a technique of storing the number of references, pointers, or handles to a resource such as an object, block of memory, disk space or other resource.**"

- Wikipedia

What do you have?

Memory management in Python

- It involves a **private heap** containing all Python objects and data structures.
- The management of this private heap is ensured internally by the Python memory manager.

Thus, we have that the Python Interpreter

- Uses what is called a reference count.

A Reference Count?

“**In computer science, reference counting is a technique of storing the number of references, pointers, or handles to a resource such as an object, block of memory, disk space or other resource.**”

- Wikipedia

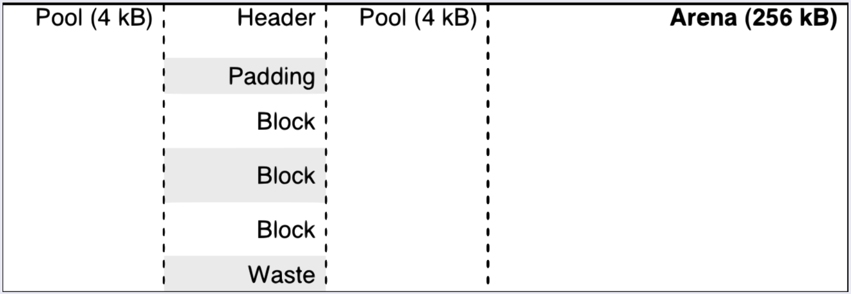
This allows

To Python to determine when

The memory occupied by an object can be reclaimed.

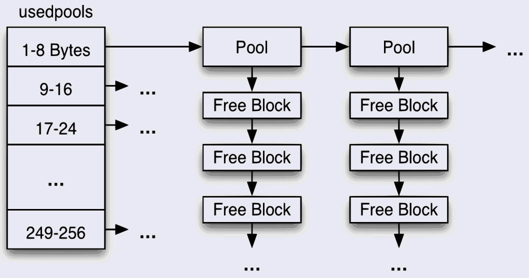
The pymalloc Allocator

It has memory allocated in the following chunks of 256 Kb called Arenas



Allocating Space for Python Objects

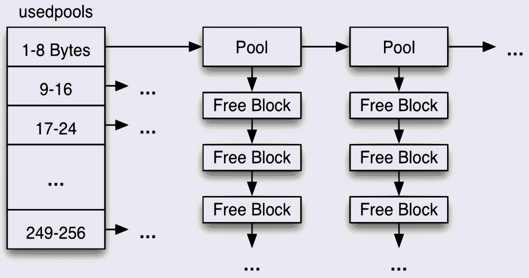
There is a list of free blocks



1. If there is a free element's pool, we pop a free block off of its list
2. If the pool is full, the pool is popped off out of the available free block
3. If there are no pools of the correct size, we need to find an available pool

Allocating Space for Python Objects

There is a list of free blocks

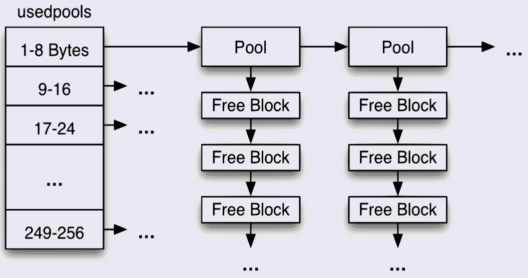


Instantiation of an Object

1. Each of the pools has a **singly linked** list of available blocks.
2. If there is a free element's pool, we pop a **free block** off of its list.
3. If the pool is full, the pool is popped off out of the available free block.
4. **If there are no pools of the correct size**, we need to find an available pool.
    - The process looks for the **freepools** linked list

Allocating Space for Python Objects

There is a list of free blocks

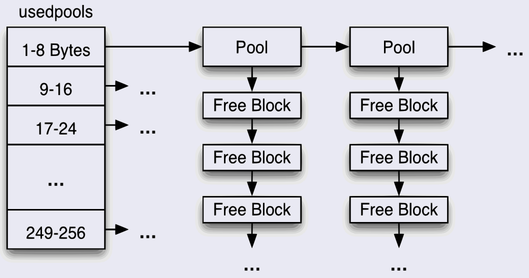


Instantiation of an Object

1. Each of the pools has a **singly linked** list of available blocks.
2. If there is a free element's pool, we pop a **free block** off of its list.
3. If the pool is full, the pool is popped off out of the available free block.
4. **If there are no pools of the correct size**, we need to find an available pool.
   - The program looks for the freepools linked list

Allocating Space for Python Objects

There is a list of free blocks

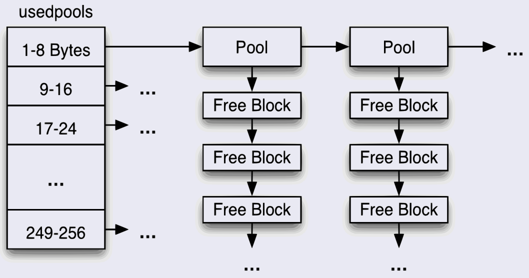


Instantiation of an Object

1. Each of the pools has a **singly linked** list of available blocks.
2. If there is a free element's pool, we pop a **free block** off of its list.
3. If the pool is full, the pool is popped off out of the available free block.
4. If there are no pools of the correct size, we need to find an available pool.

Allocating Space for Python Objects

There is a list of free blocks

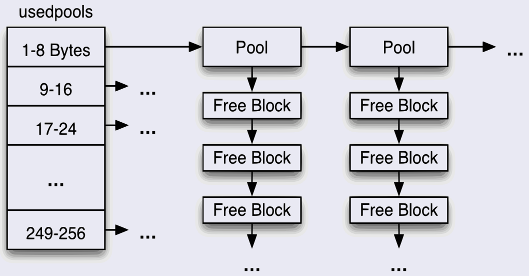


Instantiation of an Object

1. Each of the pools has a **singly linked** list of available blocks.
2. If there is a free element's pool, we pop a **free block** off of its list.
3. If the pool is full, the pool is popped off out of the available free block.
4. **If there are no pools of the correct size**, we need to find an available pool.

Allocating Space for Python Objects

There is a list of free blocks



Instantiation of an Object

1. Each of the pools has a **singly linked** list of available blocks.
2. If there is a free element's pool, we pop a **free block** off of its list.
3. If the pool is full, the pool is popped off out of the available free block.
4. **If there are no pools of the correct size**, we need to find an available pool.
   1. The process looks for the **freepools** linked list

Otherwise

We need to create a new free pool

If there is enough space in the last allocated arena.

Otherwise

We need to create a new free pool

If there is enough space in the last allocated arena.

If there is no space

We allocate a new arena using malloc() from CPython implementation.

Outline

Main Ideas of CPython's Garbage Collector

Maintain reference count.

- For every object, there is a count of the total number of references to that object.
- If that count ever falls to 0, then it deallocate that object because it is no longer alive.

- Deallocating when the reference count falls to 0 does not work for all cases.

- This is called a reference cycle.

Main Ideas of CPython's Garbage Collector

Maintain reference count.

- For every object, there is a count of the total number of references to that object.
- If that count ever falls to 0, then it deallocate that object because it is no longer alive.

However

- Periodically detect reference cycles.
- Deallocating when the reference count falls to 0 does not work for all cases.

- This is called a reference cycle.

Main Ideas of CPython's Garbage Collector

Maintain reference count.

- For every object, there is a count of the total number of references to that object.
- If that count ever falls to 0, then it deallocate that object because it is no longer alive.

However

- Periodically detect reference cycles.
- Deallocating when the reference count falls to 0 does not work for all cases.

However

- Consider two objects A and B, where A holds a reference to B and B holds a reference to A.
- This is called a reference cycle.

Main Ideas of CPython's Garbage Collector

Maintain reference count.

- For every object, there is a count of the total number of references to that object.
- If that count ever falls to 0, then it deallocate that object because it is no longer alive.

However

- Periodically detect reference cycles.
- Deallocating when the reference count falls to 0 does not work for all cases.

However

- Consider two objects A and B, where A holds a reference to B and B holds a reference to A.
- This is called a reference cycle.

Main Ideas of CPython's Garbage Collector

Maintain reference count.

- For every object, there is a count of the total number of references to that object.
- If that count ever falls to 0, then it deallocate that object because it is no longer alive.

However

- Periodically detect reference cycles.
- Deallocating when the reference count falls to 0 does not work for all cases.

However

- Consider two objects A and B, where A holds a reference to B and B holds a reference to A.
- This is called a reference cycle.

Main Ideas of CPython's Garbage Collector

Maintain reference count.

- For every object, there is a count of the total number of references to that object.
- If that count ever falls to 0, then it deallocate that object because it is no longer alive.

However

- Periodically detect reference cycles.
- Deallocating when the reference count falls to 0 does not work for all cases.

However

- Consider two objects A and B, where A holds a reference to B and B holds a reference to A.
- This is called a reference cycle.

It is called reference cycle

Therefore

- It could be the case that these are no longer live and so that both A and B should be garbage collected.
- However, this is not possible because the reference has not gone to zero!!!

- One of the classic ones is the Mark and Sweep collector

- Thus every time an entire arena was in partially_allocated arenas the arena can be released

It is called reference cycle

Therefore

- It could be the case that these are no longer live and so that both A and B should be garbage collected.
- However, this is not possible because the reference has not gone to zero!!!

Then

- Performance is enhanced with heuristics and algorithms to avoid this problems.
- One of the classic ones is the Mark and Sweep collector.

- Thus every time an entire arena was in partially_allocated_arenas, the arena can be released.

It is called reference cycle

Therefore

- It could be the case that these are no longer live and so that both A and B should be garbage collected.
- However, this is not possible because the reference has not gone to zero!!!

Then

- Performance is enhanced with heuristics and algorithms to avoid this problems.
- One of the classic ones is the Mark and Sweep collector.

In addition from Python 2.5

- Evan Jones added a belonging Arena label for each pool in the freepools (Renamed **partially_allocated_arenas**).
- Thus every time an entire arena was in **partially_allocated_arenas**, the arena can be released.

It is called reference cycle

Therefore

- It could be the case that these are no longer live and so that both A and B should be garbage collected.
- However, this is not possible because the reference has not gone to zero!!!

Then

- Performance is enhanced with heuristics and algorithms to avoid this problems.
- One of the classic ones is the Mark and Sweep collector.

In addition from Python 2.5

- Evan Jones added a belonging Arena label for each pool in the freepools (Renamed **partially_allocated_arenas**).
- Thus every time an entire arena was in **partially_allocated_arenas**, the arena can be released.

It is called reference cycle

Therefore

- It could be the case that these are no longer live and so that both A and B should be garbage collected.
- However, this is not possible because the reference has not gone to zero!!!

Then

- Performance is enhanced with heuristics and algorithms to avoid this problems.
- One of the classic ones is the Mark and Sweep collector.

In addition from Python 2.5

- Evan Jones added a belonging Arena label for each pool in the freepools (Renamed **partially_allocated_arenas**).
- Thus every time an entire arena was in **partially_allocated_arenas**, the arena can be released.

It is called reference cycle

Therefore

- It could be the case that these are no longer live and so that both A and B should be garbage collected.
- However, this is not possible because the reference has not gone to zero!!!

Then

- Performance is enhanced with heuristics and algorithms to avoid this problems.
- One of the classic ones is the Mark and Sweep collector.

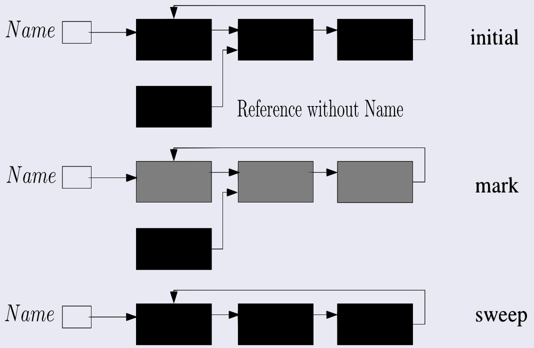
In addition from Python 2.5

- Evan Jones added a belonging Arena label for each pool in the freepools (Renamed **partially_allocated_arenas**).
- Thus every time an entire arena was in **partially_allocated_arenas**, the arena can be released.

Outline

Mark and Sweep

What the do we have?



Outline

We are done with the theoretical part

We are ready for the real stuff!!!

Now, we are ready for the practical stuff!!!