Introduction to Programming (C/C++)

08: Potpourri

Huanchen Zhang

清華大學
Tsinghua University

交叉信息研究院
Institute for Interdisciplinary Information Sciences

Agenda

- ➜ Move Semantics (Review)
- ➜ Smart Pointers
- ➜ Type Casting
- ➜ Exceptions
- ➜ Python!

Agenda

- ➜ Move Semantics (Review)
- ➜ **Smart Pointers**
- ➜ Type Casting
- ➜ Exceptions
- ➜ Python!

Smart Pointers

```
void Func(int x) {
  int *ptr = new Object(x);
  ...
  delete ptr;
}
```

Smart Pointers

```
void Func(int x) {
  int *ptr = new Object(x);
  ...
  if (x == 0) return;
  ...
  if (x < 0) return;
  ...
  delete ptr;
}
```

Smart Pointers

```
void Func(int x) {
  int *ptr = new Object(x);
  ...
  if (x == 0) return;
  ...
  if (x < 0) return;
  ...
  delete ptr;
}
```



```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  ...
  if (x == 0) return;  ⟵
  ...
  if (x < 0) return;   ⟵
  ...
  delete ptr;
}
```

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  ...
  if (x == 0) return;
  ...
  if (x < 0) return;
  ...

}
```

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  ...
  if (x == 0) return;
  ...
  if (x < 0) return;
  ...

}
```

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Resource Acquisition Is Initialization (RAII)

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  Pointer<int> ptr2 = ptr;
  ...
}
```

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  Pointer<int> ptr2 = ptr;
  ...
}
```

Double Free

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  Pointer<int> ptr2 = ptr;
  ...
}
```

Double Free

1. Disable copy semantics, only allow move semantics

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  Pointer<int> ptr2 = ptr;
  ...
}
```

Double Free

1. Disable copy semantics, only allow move semantics

   `std::unique_ptr`

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  Pointer<int> ptr2 = ptr;
  ...
}
```

Double Free

1. Disable copy semantics, only allow move semantics

   `std::unique_ptr`

2. Add reference counter

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

Smart Pointers

```
void Func(int x) {
  Pointer<int> ptr(new Object(x));
  Pointer<int> ptr2 = ptr;
  ...
}
```

Double Free

1. Disable copy semantics, only allow move semantics

   `std::unique_ptr`

2. Add reference counter

   `std::shared_ptr`

```
template <typename T>
class Pointer {
 public:
  Pointer(T *ptr) : ptr_(ptr) {}
  ~Pointer() { delete ptr_; }
  T &operator*() const { return *ptr_; }
  T *operator->() const { return ptr_; }

 private:
  T *ptr_;
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);




}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);
  std::unique_ptr<Data> d1_ptr(new Data(10000));



}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);
  std::unique_ptr<Data> d1_ptr(new Data(10000));
  std::unique_ptr<Data> d2_ptr;

}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);
  std::unique_ptr<Data> d1_ptr(new Data(10000));
  std::unique_ptr<Data> d2_ptr;
  d2_ptr = d1_ptr;
}
```

std::unique_ptr

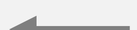
```cpp
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);
  std::unique_ptr<Data> d1_ptr(new Data(10000));
  std::unique_ptr<Data> d2_ptr;
  d2_ptr = d1_ptr; ☒
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);
  std::unique_ptr<Data> d1_ptr(new Data(10000));
  std::unique_ptr<Data> d2_ptr;
  d2_ptr = std::move(d1_ptr);
}
```

std::unique_ptr

```cpp
#include <memory>
#include <utility>

int main() {
  std::unique_ptr<int> ptr(new int[1000]);
  std::unique_ptr<Data> d1_ptr(new Data(10000));
  std::unique_ptr<Data> d2_ptr;
  d2_ptr = std::move(d1_ptr);
}
```



Ownership transferred

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  Data *d = new Data(10000);
  std::unique_ptr<Data> d1_ptr(d);
  std::unique_ptr<Data> d2_ptr(d);
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  Data *d = new Data(10000);
  std::unique_ptr<Data> d1_ptr(d);
  std::unique_ptr<Data> d2_ptr(d);
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  Data *d = new Data(10000);
  std::unique_ptr<Data> d1_ptr(d);
  delete d;
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  Data *d = new Data(10000);
  std::unique_ptr<Data> d1_ptr(d);
  delete d;
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  auto d1_ptr(std::make_unique<Data>(10000));
  ...
}
```

std::unique_ptr

```
#include <memory>
#include <utility>

int main() {
  auto d1_ptr(std::make_unique<Data>(10000));
  ...
}
```

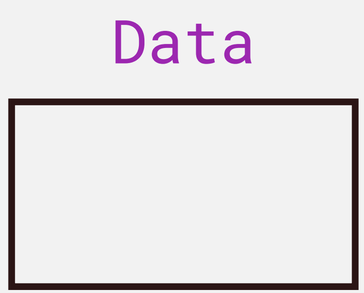
- Avoid using **new/delete**
- Avoid having **raw pointers**

std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d1_ptr);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```

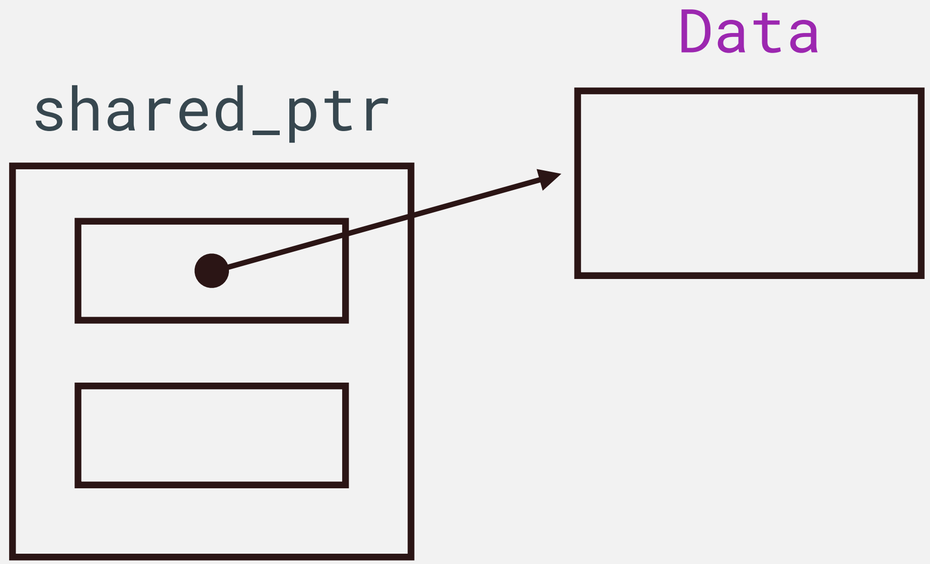
std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d1_ptr);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



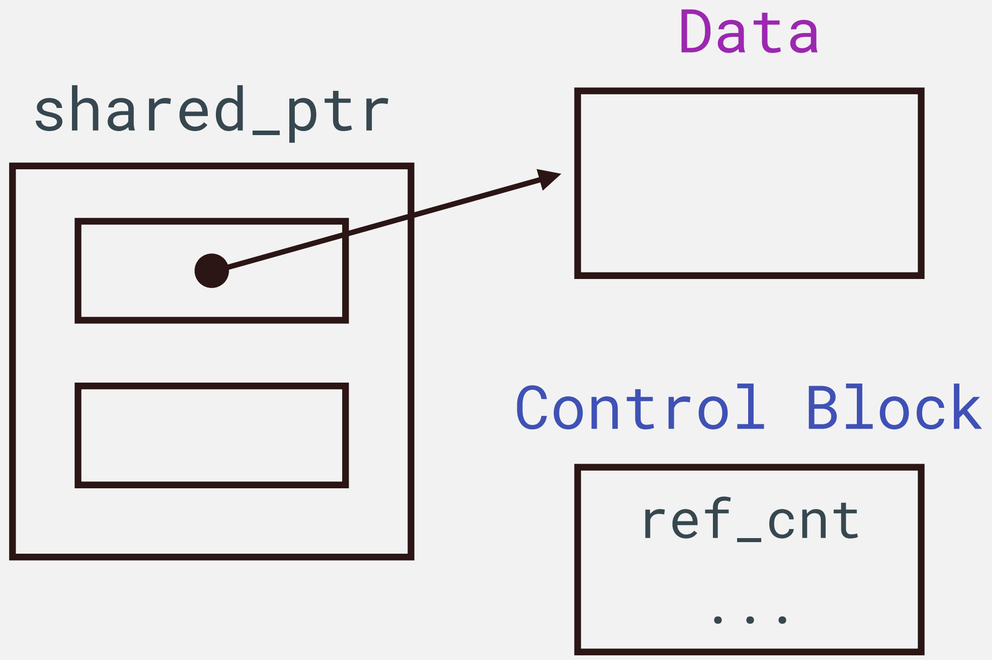
std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d1_ptr);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



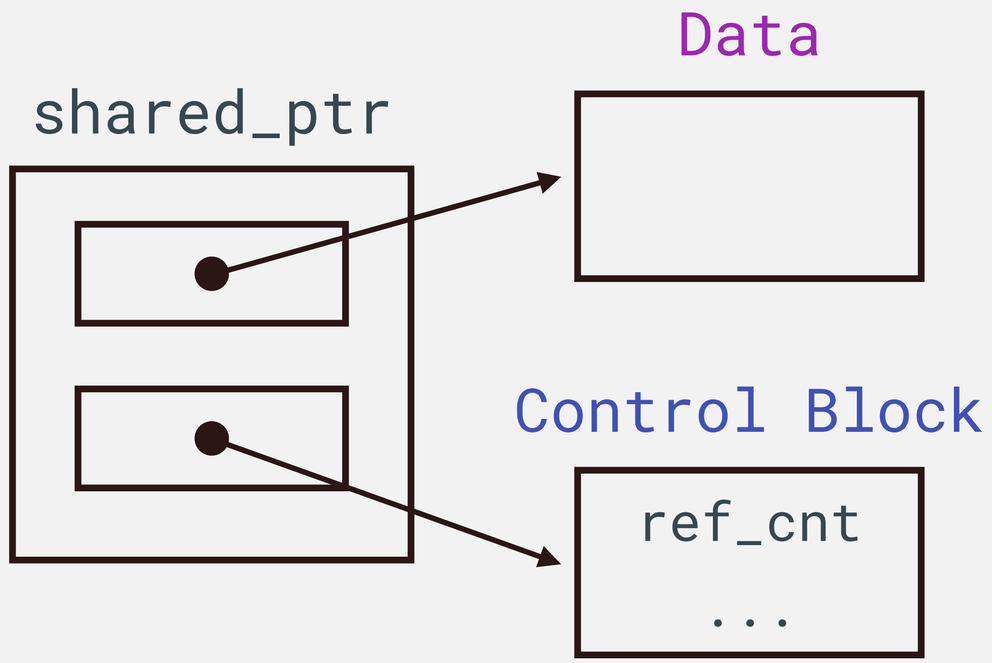
std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d1_ptr);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



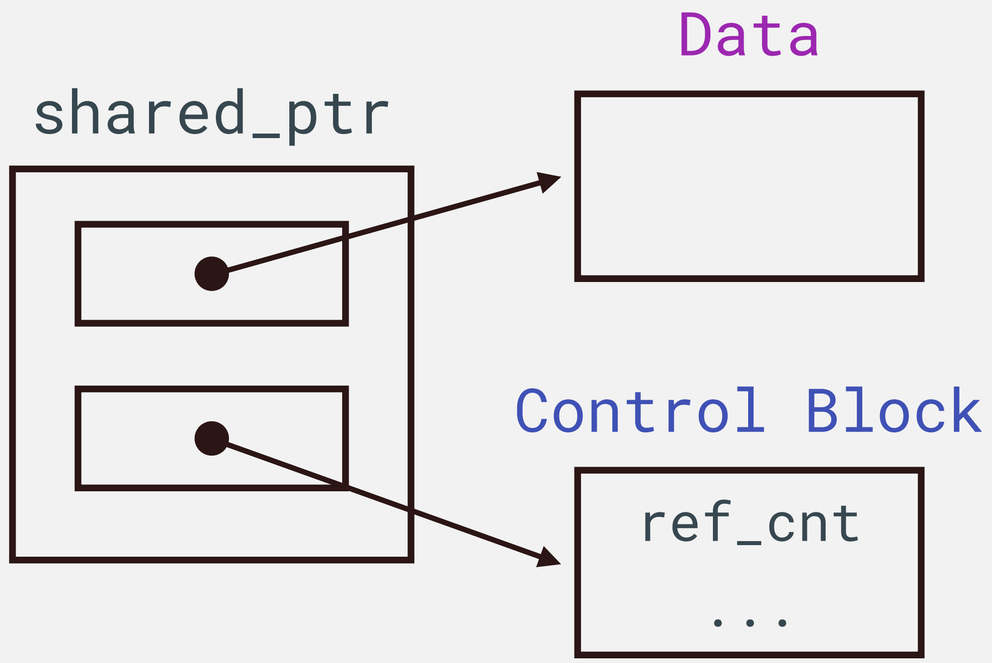
std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d1_ptr);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



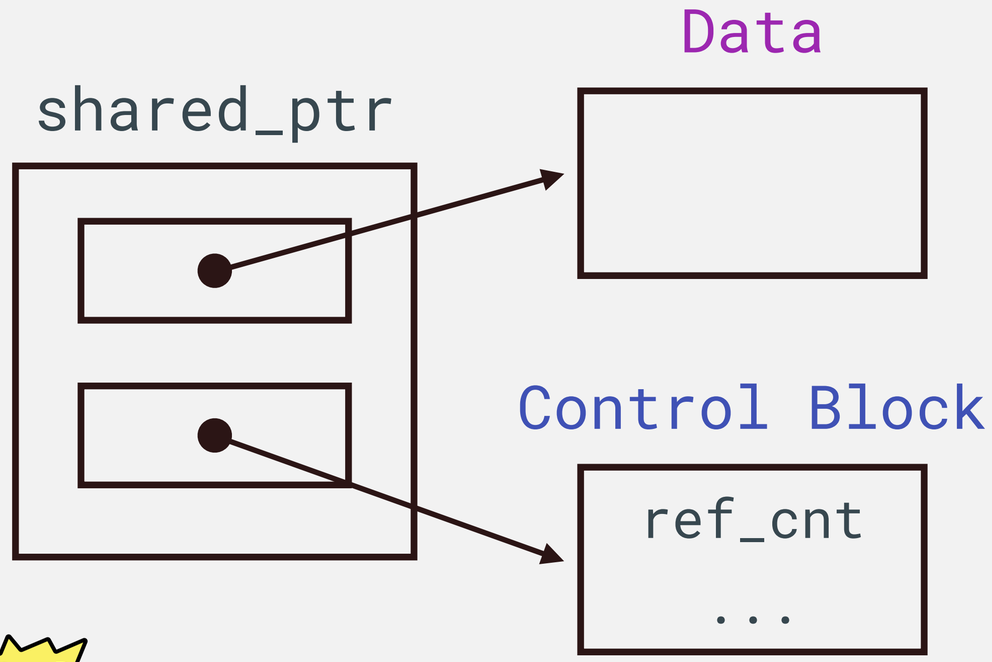
std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



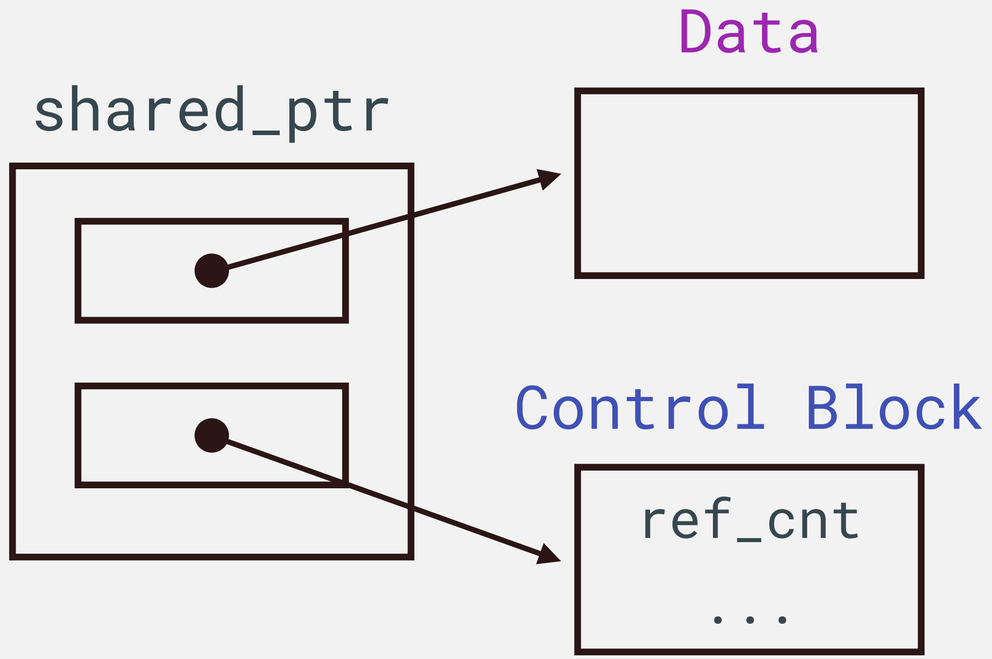
std::shared_ptr

```
#include <memory>

int main() {
  Data *d = new Data(100);
  std::shared_ptr<Data> d1_ptr(d);
  {
    std::shared_ptr<Data> d2_ptr(d);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



std::shared_ptr

```
#include <memory>

int main() {
  auto d1_ptr(std::make_shared<Data>(100));
  {
    auto d2_ptr(d1_ptr);
  }
  std::cout << d1_ptr->GetSize() << “\n”;
}
```



Agenda

- ➜ Move Semantics (Review)
- ➜ Smart Pointers
- ➜ **Type Casting**
- ➜ Exceptions
- ➜ Python!

Type Casting in C++

- C-Style Cast
- Static Cast
- Dynamic Cast
- Const Cast
- Reinterpret Cast

Type Casting in C++

- C-Style Cast
- Static Cast
- Dynamic Cast
- Const Cast
- Reinterpret Cast

Named Cast

C-Style Cast (1)

```
int x = 5;
int y = 2;
double d = x / y;

std::cout << d << “\n”;
```

C-Style Cast (1)

```
int x = 5;
int y = 2;
double d = x / y;

std::cout << d << “\n”;
```

```
>>> 2
```

C-Style Cast (1)

```
int x = 5;
int y = 2;
double d = (double)x / y;

std::cout << d << “\n”;
```

C-Style Cast (1)

```
int x = 5;
int y = 2;
double d = (double)x / y;

std::cout << d << “\n”;
```

```
>>> 2.5
```

C-Style Cast (1)

```
int x = 5;
int y = 2;
double d = double(x) / y;

std::cout << d << "\n";
```

```
>>> 2.5
```

Static Cast

```
int x = 5;
int y = 2;
double d = static_cast<double>(x) / y;

std::cout << d << “\n”;
```

```
>>> 2.5
```

Static Cast

```
int x = 5;
int y = 2;
double d = static_cast<double>(x) / y;

std::cout << d << “\n”;
```

```
>>> 2.5
```

➜ Best used to convert one primitive type to another

Static Cast

```
int x = 5;
int y = 2;
double d = static_cast<double>(x) / y;

std::cout << d << “\n”;
```

```
>>> 2.5
```

➜ Best used to convert one primitive type to another

➜ Provides compile-time type checking

Static Cast

```
int x = 5;
int y = 2;
int d = static_cast<int>(“Hello”);
```

- ➜ Best used to convert one primitive type to another
- ➜ Provides compile-time type checking

Static Cast

```
int x = 5;
int y = 2;
int d = static_cast<int>(“Hello”);
```

- ➜ Best used to convert one primitive type to another
- ➜ Provides compile-time type checking

C-Style Cast (2)

```
const int x = 5;
int y = 2;
int &r = (int &)x;
```

C-Style Cast (2)

```
const int x = 5;
int y = 2;
int &r = (int &)x;
```

⟵ C-style cast allows casting away const

C-Style Cast (2)

```
const int x = 5;
int y = 2;
int &r = static_cast<int &>(x);
```

Const Cast 

```
const int x = 5;
int y = 2;
int &r = const_cast<int &>(x);
```

➜ Used to remove or add const to a variable

C-Style Cast (3)

```
typedef struct Tuple {
  int x;
  int y;
}tuple;

int main() {
  int *ptr = new int[2];
  ptr[0] = 3;
  ptr[1] = 5;
  tuple *t_ptr = (tuple *)ptr;
  std::cout << t_ptr->x << “\n” << t_ptr->y << “\n”;
  delete[] ptr;
}
```

C-Style Cast (3)

```
typedef struct Tuple {
  int x;
  int y;
}tuple;

int main() {
  int *ptr = new int[2];
  ptr[0] = 3;
  ptr[1] = 5;
  tuple *t_ptr = (tuple *)ptr;
  std::cout << t_ptr->x << “\n” << t_ptr->y << “\n”;
  delete[] ptr;
}
```

➜ **Convert pointer types**

- reinterpret the bits in the pointed memory region

Reinterpret Cast 

```
typedef struct Tuple {
  int x;
  int y;
}tuple;

int main() {
  int *ptr = new int[2];
  ptr[0] = 3;
  ptr[1] = 5;
  tuple *t_ptr = reinterpret_cast<tuple *>(ptr);
  std::cout << t_ptr->x << “\n” << t_ptr->y << “\n”;
  delete[] ptr;
}
```

➜ **Convert pointer types**

- reinterpret the bits in the pointed memory region

Motivation for Dynamic Cast

```
class Base {
 public:
  Base(int id) : id_{id} {};
 protected:
  int id_;
};

class Derived : public Base {
 public:
  Derived(int id, const std::string &name)
    : Base(id), name_(name) {};
  const std::string &GetName() const { return name_; }
 private:
  std::string name_;
};
```

```
Base *MakeObject(int t) {
  if (t == 0) return new Base(1);
  else return new Derived(2, “Hello”);
}

int main() {
  Base *ptr = MakeObject(1);
  ptr->GetName();
  delete ptr;
}
```

Motivation for Dynamic Cast

```
class Base {
 public:
  Base(int id) : id_{id} {};
 protected:
  int id_;
};

class Derived : public Base {
 public:
  Derived(int id, const std::string &name)
    : Base(id), name_(name) {};
  const std::string &GetName() const { return name_; }
 private:
  std::string name_;
};
```

```
Base *MakeObject(int t) {
  if (t == 0) return new Base(1);
  else return new Derived(2, “Hello”);
}

int main() {
  Base *ptr = MakeObject(1);
  ptr->GetName(); ✗
  delete ptr;
}
```

Motivation for Dynamic Cast

```
class Base {
 public:
  Base(int id) : id_{id} {};
 protected:
  int id_;
};

class Derived : public Base {
 public:
  Derived(int id, const std::string &name)
    : Base(id), name_(name) {};
  const std::string &GetName() const { return name_; }
 private:
  std::string name_;
};
```

```
Base *MakeObject(int t) {
  if (t == 0) return new Base(1);
  else return new Derived(2, “Hello”);
}

int main() {
  Base *ptr = MakeObject(1);
  static_cast<Derived *>(ptr)->GetName();
  delete ptr;
}
```

Motivation for Dynamic Cast

```
class Base {
 public:
  Base(int id) : id_{id} {};
 protected:
  int id_;
};

class Derived : public Base {
 public:
  Derived(int id, const std::string &name)
    : Base(id), name_(name) {};
  const std::string &GetName() const { return name_; }
 private:
  std::string name_;
};
```

```
Base *MakeObject(int t) {
  if (t == 0) return new Base(1);
  else return new Derived(2, “Hello”);
}

int main() {
  Base *ptr = MakeObject(1);
  static_cast<Derived *>(ptr)->GetName();
  delete ptr;
}
```

Motivation for Dynamic Cast

```
class Base {
 public:
  Base(int id) : id_{id} {};
 protected:
  int id_;
};

class Derived : public Base {
 public:
  Derived(int id, const std::string &name)
     : Base(id), name_(name) {};
  const std::string &GetName() const { return name_; }
 private:
  std::string name_;
};
```

```
Base *MakeObject(int t) {
  if (t == 0) return new Base(1);
  else return new Derived(2, “Hello”);
}

int main() {
  Base *ptr = MakeObject(1);
  dynamic_cast<Derived *>(ptr)->GetName();
  delete ptr;
}
```

Dynamic Cast

```
Base *MakeObject(int t) {
  if (t == 0)
    return new Base(1);
  else
    return new Derived(2, “Hello”);
}

int main() {
  Base *b_ptr = MakeObject(1);
  Derived *d_ptr = dynamic_cast<Derived *>(b_ptr);
  std::cout << d_ptr->GetName() << “\n”;
  delete ptr;
}
```

➜ Convert base-class pointer to derived-class pointer

Dynamic Cast

```
Base *MakeObject(int t) {
  if (t == 0)
    return new Base(1);
  else
    return new Derived(2, “Hello”);
}

int main() {
  Base *b_ptr = MakeObject(1);
  Derived *d_ptr = dynamic_cast<Derived *>(b_ptr);
  std::cout << d_ptr->GetName() << “\n”;
  delete ptr;
}
```

➜ Convert base-class pointer to derived-class pointer

← **upcast**

← **downcast**

Dynamic Cast

```
Base *MakeObject(int t) {
  if (t == 0)
    return new Base(1);
  else
    return new Derived(2, “Hello”);
}

int main() {
  Irrelevant *b_ptr = new Irrelevant();
  Derived *d_ptr = dynamic_cast<Derived *>(b_ptr);
  std::cout << d_ptr->GetName() << “\n”;
  delete ptr;
}
```

➜ Convert base-class pointer to derived-class pointer

Dynamic Cast

```
Base *MakeObject(int t) {
  if (t == 0)
    return new Base(1);
  else
    return new Derived(2, “Hello”);
}

int main() {
  Irrelevant *b_ptr = new Irrelevant();
  Derived *d_ptr = dynamic_cast<Derived *>(b_ptr);
  std::cout << d_ptr->GetName() << “\n”;
  delete ptr;
}
```

➜ Convert base-class pointer to derived-class pointer

dynamic_cast returns nullptr

Dynamic Cast

```
Base *MakeObject(int t) {
  if (t == 0)
    return new Base(1);
  else
    return new Derived(2, “Hello”);
}

int main() {
  Irrelevant *b_ptr = new Irrelevant();
  Derived *d_ptr = dynamic_cast<Derived *>(b_ptr);
  if (d_ptr)
    std::cout << d_ptr->GetName() << “\n”;
  delete ptr;
}
```

➜ Convert base-class pointer to derived-class pointer

Dynamic Cast

```
Base *MakeObject(int t) {
  if (t == 0)
    return new Base(1);
  else
    return new Derived(2, “Hello”);
}
```

- ➜ Convert base-class pointer to derived-class pointer
- ➜ Always check for null pointer after dynamic cast

```
int main() {
  Irrelevant *b_ptr = new Irrelevant();
  Derived *d_ptr = dynamic_cast<Derived *>(b_ptr);
  if (d_ptr)
    std::cout << d_ptr->GetName() << “\n”;
  delete ptr;
}
```

C++ Type Casting Summary

➜ `static_cast`

- most common type casting, mainly used for conversion between primitive types, can be implicit and explicit

➜ `dynamic_cast`

- for handling polymorphism, base-class pointer —> derived-class pointer (downcast)

➜ `const_cast`

- for removing or add const to a variable. **Avoid unless you have specific reasons.**

➜ `reinterpret_cast`

- for converting pointer types. **Avoid unless you have specific reasons.**

➜ C-Style Cast **Avoid**

Agenda

- ➜ Move Semantics (Review)
- ➜ Smart Pointers
- ➜ Type Casting
- ➜ **Exceptions**
- ➜ Python!

Why Exceptions?

➜ Return code is not enough for handling errors

```
int divide(int x, int y) {
  if (y == 0)
    return -1;
  double z = static_cast<double>(x) / y;
  std::cout << z << “\n”;
  return 0;
}
```

Why Exceptions?

➜ Return code is not enough for handling errors

```
double divide(int x, int y) {
  return static_cast<double>(x) / y;
}
```

Why Exceptions?

➜ Return code is not enough for handling errors

```
double divide(int x, int y, int &success) {
  if (y == 0) {
    success = -1;
    return 0.0;
  }
  success = 0;
  return static_cast<double>(x) / y;
}
```

Why Exceptions?

➜ Return code is not enough for handling errors

- Function already has a return value
- What if something inside a constructor goes wrong?
- What if the immediate caller must propagate the error up the call stack?

▪ ▪ ▪

Why Exceptions?

➜ Return code is not enough for handling errors

- Function already has a return value
- What if something inside a constructor goes wrong?
- What if the immediate caller must propagate the error up the call stack?

▪ ▪ ▪

➜ **Exception**

- A mechanism that decouples error handling from normal control flow

Raise Exceptions

```
throw -1;
throw INVALID_INDEX;
throw “divide by zero”;
throw MyException(“divide by zero”);
```

Observe Exceptions

```
try {
  throw -1;
}
```

```
try {
  if (x < 0)
    throw “cannot take sqrt of a negative number”;
  std::cout << std::sqrt(x) << “\n”;
}
```

Handle Exceptions

```
catch (int e) {
  std::cerr << “caught int exception ” << e << “\n”;
}

catch (const std::string&) {
  std::cerr << “caught string exception\n”;
}
```

Put It Together

```
try {
  throw -1;
}
catch (int x) {
  std::cerr << “int error ” << x << “\n”;
}
catch (const std::string&) {
  std::cerr << “string error\n”;
}
catch (const MyException&) {
  std::cerr << “MyException error\n”;
}
```

Put It Together

```
int x = 0;
std::cin >> x;
try {
  if (x < 0)
    throw “cannot take sqrt of a negative number”;
  std::cout << std::sqrt(x) << “\n”;
}
catch (const char *e) {
  std::cerr << “Error: ” << e << “\n”;
}
std::cout << (x * x) << “\n”;
```

Exception and Functions

```
double MySqrt(double x) {
  if (x < 0)
    throw “cannot take sqrt of a negative number”;
  return std::sqrt(x);
}
int main() {
  int x = 0;
  std::cin >> x;
  try {
    std::cout << MySqrt(x) << “\n”;
  }
  catch (const char *e) {
    std::cerr << “Error: ” << e << “\n”;
  }
}
```

Exception and Functions

```
double MySqrt(double x) {
  if (x < 0)
    throw “cannot take sqrt of a negative number”;
  return std::sqrt(x);
}
int main() {
  int x = 0;
  std::cin >> x;
  try {
    std::cout << MySqrt(x) << “\n”;
  }
  catch (const char *e) {
    std::cerr << “Error: ” << e << “\n”;
  }
}
```

catches exceptions from the called functions

Exception and Functions

```
double MySqrt(double x) {
  if (x < 0)
    throw “cannot take sqrt of a negative number”;
  return std::sqrt(x);
}
int main() {
  int x = 0;
  std::cin >> x;
  try {
    std::cout << MySqrt(x) << “\n”;
  }
  catch (const char *e) {
    std::cerr << “Error: ” << e << “\n”;
  }
}
```

➜ Applications may want to handle errors differently

⟵ catches exceptions from the called functions

Stack Unwinding

```
void D() {
  std::cout << “D Start\n”;
  throw -1;
  std::cout << “D End\n”;
}
void C() {
  std::cout << “C Start\n”;
  D();
  std::cout << “C End\n”;
}
void B() {
  std::cout << “B Start\n”;
  try { C(); } catch (double) {  ①  }
  try { } catch (int) {  ②  }
  std::cout << “B End\n”;
}
```

```
void A() {
  std::cout << “A Start\n”;
  try { B(); } catch (int) {  ③  }
  std::cout << “A End\n”;
}
int main {
  std::cout << “main Start\n”;
  try { A() } catch (int) {  ④  }
  std::cout << “main End\n”;
}
```

Stack Unwinding

```
void D() {
  std::cout << “D Start\n”;
  throw -1;
  std::cout << “D End\n”;
}
void C() {
  std::cout << “C Start\n”;
  D();
  std::cout << “C End\n”;
}
void B() {
  std::cout << “B Start\n”;
  try { C(); } catch (double) { }   ①
  try { } catch (int) { }   ②
  std::cout << “B End\n”;
}
```

```
void A() {
  std::cout << “A Start\n”;
  try { B(); } catch (int) { }   ③
  std::cout << “A End\n”;
}
int main {
  std::cout << “main Start\n”;
  try { A() } catch (int) { }   ④
  std::cout << “main End\n”;
}
```

```
>>> main Start
>>> A Start
>>> B Start
>>> C Start
>>> D Start
```

Stack Unwinding

```
void D() {
  std::cout << “D Start\n”;
  throw -1;
  std::cout << “D End\n”;
}
void C() {
  std::cout << “C Start\n”;
  D();
  std::cout << “C End\n”;
}
void B() {
  std::cout << “B Start\n”;
  try { C(); } catch (double) {  ①  }
  try { } catch (int) {  ②  }
  std::cout << “B End\n”;
}
```

```
void A() {
  std::cout << “A Start\n”;
  try { B(); } catch (int) {  ③  }
  std::cout << “A End\n”;
}
int main {
  std::cout << “main Start\n”;
  try { A() } catch (int) {  ④  }
  std::cout << “main End\n”;
}
```

```
>>> main Start
>>> A Start
>>> B Start
>>> C Start
>>> D Start
```

Stack Unwinding

```
void D() {
  std::cout << “D Start\n”;
  throw -1;
  std::cout << “D End\n”;
}
void C() {
  std::cout << “C Start\n”;
  D();
  std::cout << “C End\n”;
}
void B() {
  std::cout << “B Start\n”;
  try { C(); } catch (double) {  ①  }
  try { } catch (int) {  ②  }
  std::cout << “B End\n”;
}
```

```
void A() {
  std::cout << “A Start\n”;
  try { B(); } catch (int) {  ③  }
  std::cout << “A End\n”;
}
int main {
  std::cout << “main Start\n”;
  try { A() } catch (int) {  ④  }
  std::cout << “main End\n”;
}
```

```
>>> main Start
>>> A Start
>>> B Start
>>> C Start
>>> D Start
```

```
>>> A End
>>> main End
```

Exception Class

```
class IntArray {
 public:
  IntArray(int size) : size_(size) {
    if (size <= 0) throw -1;
    *data_ = new int[size];
  }
  int GetLength() const { return size_; }
  int &operator[](const int idx) {
    if (idx < 0 || idx >= GetLength())
      throw -2;
    return data_[idx];
  }
 private:
  int size_;
  int *data_;
};
```

```
int main() {
  int x = 0;
  std::cin >> x;
  try {
    IntArray a(10);
    int val = a[x];
  } catch (int) {
    ???
  }
}
```

Exception Class

```
class IntArray {
 public:
  IntArray(int size) : size_(size) {
    if (size <= 0) throw -1;
    *data_ = new int[size];
  }
  int GetLength() const { return size_; }
  int &operator[](const int idx) {
    if (idx < 0 || idx >= GetLength())
      throw -2;
    return data_[idx];
  }
 private:
  int size_;
  int *data_;
};
```

```
class ConstructorException {
 public:
  ConstructorException(const std::string &msg)
    : msg_(msg) { }
  const std::string &GetError() const {return msg_;}
 private:
  std::string msg_;
};
class ArrayIndexException {
 public:
  ArrayIndexException(const std::string &msg)
    : msg_(msg) { }
  const std::string &GetError() const {return msg_;}
 private:
  std::string msg_;
};
```

Exception Class

```
class IntArray {
 public:
  IntArray(int size) : size_(size) {
    if (size <= 0) throw ConstructorException(“construction fail”);
    *data_ = new int[size];
  }
  int GetLength() const { return size_; }
  int &operator[](const int idx) {
    if (idx < 0 || idx >= GetLength())
      throw ArrayIndexException(“index out of bound”);
    return data_[idx];
  }
 private:
  int size_;
  int *data_;
};
```

Exception Class

```
class IntArray {
 public:
  IntArray(int size) : size_(size) {
    if (size <= 0)
      throw ConstructorException(“construction fail”);
    *data_ = new int[size];}
  int GetLength() const { return size_; }
  int &operator[](const int idx) {
    if (idx < 0 || idx >= GetLength())
      throw ArrayIndexException(“index out of bound”);
    return data_[idx];
  }
 private:
  int size_;
  int *data_;
};
```

```
int main() {
  int x = 0;
  std::cin >> x;
  try {
    IntArray a(10);
    int val = a[x];
  } catch (ConstructorException e) {
    std::cout << e.GetError() << “\n”;
  } catch (ArrayIndexException e) {
    std::cout << e.GetError() << “\n”;
  }
}
```

Exceptions and Inheritance

```
class BaseException {
 public:
  BaseException() {}
};
class DerivedException : public BaseException {
 public:
  DerivedException() {}
};

int main {
  try { throw DerivedException(); }
  catch (const BaseException &b) { std::cerr << “Caught Base Exception\n”; }
  catch (const DerivedException &d) { std::cerr << “Caught Derived Exception\n”; }
}
```

Exceptions and Inheritance

```
class BaseException {
 public:
  BaseException() {}
};
class DerivedException : public BaseException {
 public:
  DerivedException() {}
};

int main {
  try { throw DerivedException(); }
  catch (const BaseException &b) { std::cerr << “Caught Base Exception\n”; }
  catch (const DerivedException &d) { std::cerr << “Caught Derived Exception\n”; }
}
>>> Caught Base Exception
```

Exceptions and Inheritance

```
class BaseException {
 public:
  BaseException() {}
};
class DerivedException : public BaseException {
 public:
  DerivedException() {}
};

int main {
  try { throw DerivedException(); }
  catch (const BaseException &b) { std::cerr << “Caught Base Exception\n”; }
  catch (const DerivedException &d) { std::cerr << “Caught Derived Exception\n”; }
}
>>> Caught Base Exception
```

is-a

Exceptions and Inheritance

```
class BaseException {
 public:
  BaseException() {}
};
class DerivedException : public BaseException {
 public:
  DerivedException() {}
};

int main {
  try { throw DerivedException(); }
  catch (const BaseException &b) { std::cerr << “Caught Base Exception\n”; }
  catch (const DerivedException &d) { std::cerr << “Caught Derived Exception\n”; }
}
>>> Caught Base Exception
```

⟵ **is-a**

Exceptions and Inheritance

```
class BaseException {
 public:
  BaseException() {}
};
class DerivedException : public BaseException {
 public:
  DerivedException() {}
};

int main {
  try { throw DerivedException(); }
  catch (const DerivedException &b) { std::cerr << “Caught Derived Exception\n”; }
  catch (const BaseException &d) { std::cerr << “Caught Base Exception\n”; }
}
>>> Caught Derived Exception
```

is-a

Exceptions and Inheritance

```
class BaseException {
 public:
  BaseException() {}
};
class DerivedException : public BaseException {
 public:
  DerivedException() {}
};

int main {
  try { throw DerivedException(); }
  catch (const DerivedException &b) { std::cerr << “Caught Derived Exception\n”; }
  catch (const BaseException &d) { std::cerr << “Caught Base Exception\n”; }
}
>>> Caught Derived Exception
```

➜ Derived exception handler should be listed before base exception handler

⟵ **is-a**

`std::exception`

➜ Exceptions thrown in standard library are derived from **`std::exception`**

`std::exception`

➜ Exceptions thrown in standard library are derived from **`std::exception`**

- unable to allocate memory —> **`std::bad_alloc`**
- failed dynamic cast —> **`std::bad_cast`**

`...`

`std::exception`

➜ Exceptions thrown in standard library are derived from **`std::exception`**

- unable to allocate memory —> **`std::bad_alloc`**
- failed dynamic cast —> **`std::bad_cast`**

...

```
try {
  std::string s;
  s.resize(std::numeric_limits<std::size_t>::max())
} catch (const std::exception& e) {
  std::cerr << “std::exception: ” << e.what() << “\n”;
}
```

`std::exception`

➜ Exceptions thrown in standard library are derived from **`std::exception`**

- unable to allocate memory —> **`std::bad_alloc`**
- failed dynamic cast —> **`std::bad_cast`**

. . .

```
try {
  std::string s;
  s.resize(std::numeric_limits<std::size_t>::max())
} catch (const std::exception& e) {
  std::cerr << “std::exception: ” << e.what() << “\n”;
}
```

`>>> std::exception: basic_string`

Reclaim Resources

```
try {
  auto *obj = new Object(5);
  processObject(obj);
  delete obj;
}
catch (const ObjectException &e) {
  std::cerr << “Fail to process object: ” << e.what() << “\n”;
}
```

Reclaim Resources

```
try {
  auto *obj = new Object(5);
  processObject(obj);
  delete obj;
}
catch (const ObjectException &e) {
  std::cerr << “Fail to process object: ” << e.what() << “\n”;
}
```

Memory Leak!

Reclaim Resources

```
Object *obj = nullptr;
try {
  obj = new Object(5);
  processObject(obj);
}
catch (const ObjectException &e) {
  std::cerr << “Fail to process object: ” << e.what() << “\n”;
}
delete obj;
```

Reclaim Resources

```
try {
  auto *obj = new Object(5);
  std::unique_ptr<Object> obj_uptr(obj);
  processObject(obj);
}
catch (const ObjectException &e) {
  std::cerr << “Fail to process object: ” << e.what() << “\n”;
}
```

Agenda

- ➜ Move Semantics (Review)
- ➜ Smart Pointers
- ➜ Type Casting
- ➜ Exceptions
- ➜ **Python!**

C++ vs. Python: A First Glance

```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

```python
def isPrime(k):
  for j in range(2, k//2 + 1):
    if not(k % j) :
      return 0
  return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

C++ vs. Python: A First Glance

```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

```python
def isPrime(k):
  for j in range(2, k//2 + 1):
    if not(k % j) :
      return 0
  return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

C++ vs. Python: A First Glance

```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

```python
def isPrime(k):
  for j in range(2, k//2 + 1):
    if not(k % j) :
      return 0
  return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

C++ vs. Python: A First Glance

```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

```python
def isPrime(k):
    for j in range(2, k//2 + 1):
        if not(k % j):
            return 0
    return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

C++ vs. Python: A First Glance

```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

```python
def isPrime(k):
  for j in range(2, k//2 + 1):
    if not(k % j) :
      return 0
  return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

C++ vs. Python: A First Glance

```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

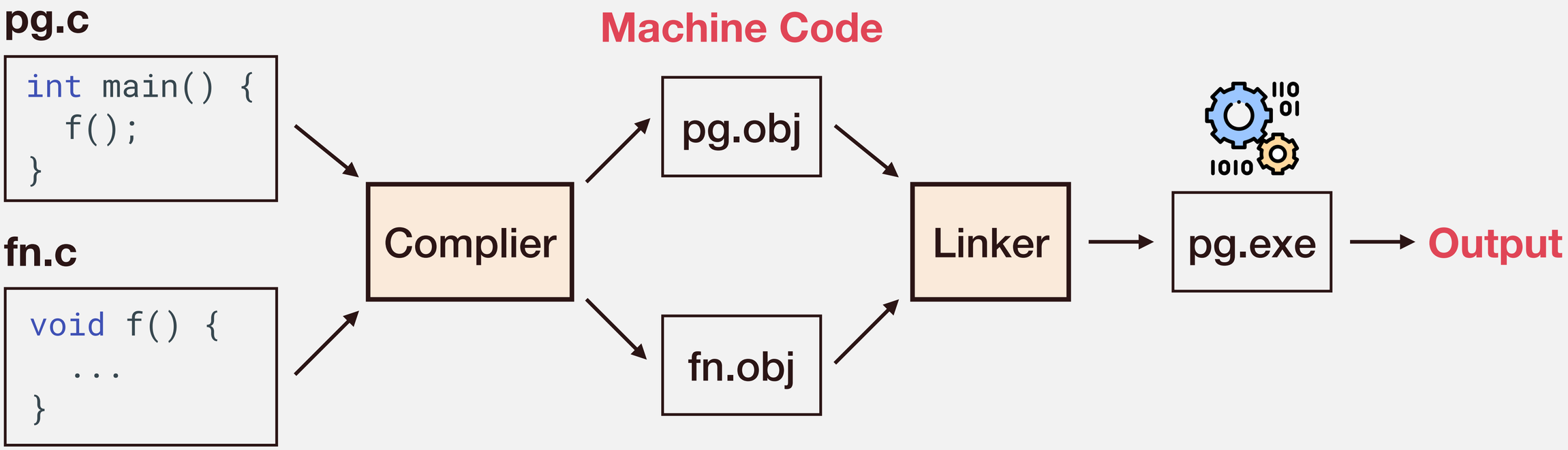
```python
def isPrime(k):
  for j in range(2, k//2 + 1):
    if not(k % j) :
      return 0
  return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

C++ vs. Python: A First Glance

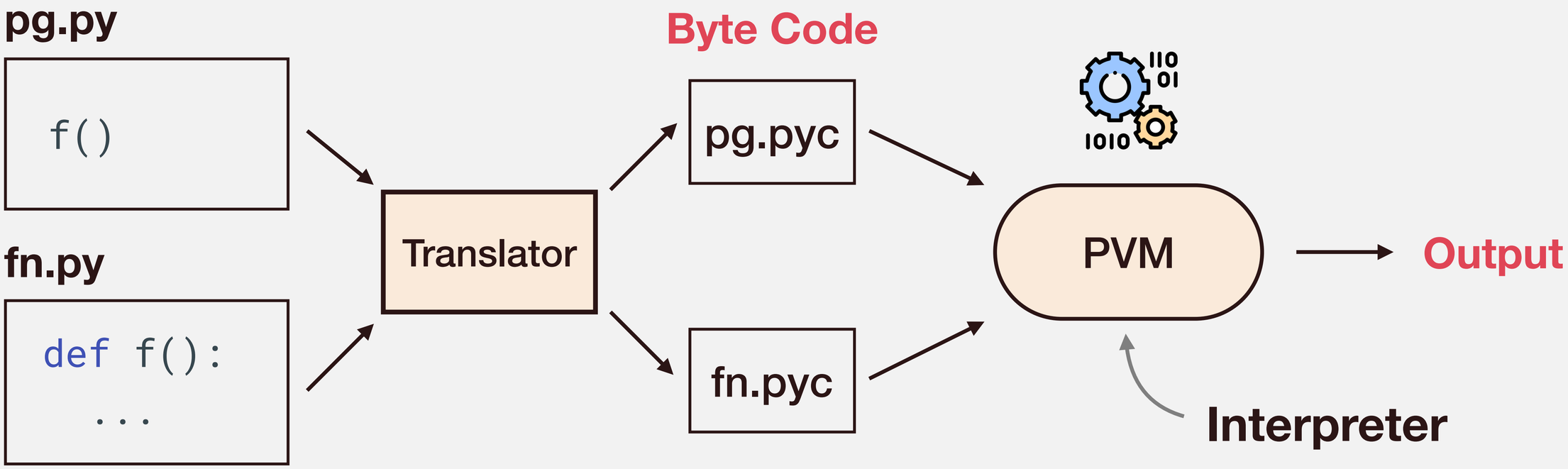
```cpp
#include <iostream>
int isPrime(int k) {
  for (int j = 2; j <= (k/2); j++) {
    if (k % j == 0)
      return 0;
  }
  return 1;
}
int main() {
  int n = 0, num_primes = 0;
  // take user input
  std::cin >> n;
  for (int i = 2; i <= n; i++)
    num_primes += isPrime(i);
  std::cout << num_primes << "\n";
}
```

```python
def isPrime(k):
  for j in range(2, k//2 + 1):
    if not(k % j) :
      return 0
  return 1

# take user input
n = int(input("Please enter a number: "))
num_primes = 0

for i in range (2, n + 1):
    num_primes += isPrime(i)

print(str(num_primes))
```

Compiled vs. Interpreted Languages



Compiled vs. Interpreted Languages



Running Python

➜ Check Python version

```
$ python -V
```

➜ Running as a script

```
$ python xxx.py
```

➜ Interactive mode

```
$ python
>>> 1 + 1
>>> exit()/quit()
```

Everything is an Object

Immutable Built-in Data Types

- `int`
  - equiv. C++ short, long, unsigned, …
- `float`
  - equiv. C++ double
- `complex`
  - complex number
  - e.g., `1 + 2j`
- `bool`
  - `True`, `False`

Immutable Built-in Data Types

- `str`
  - `'hello'` ≡ `"hello"`
  - No `char` type in Python
  - No null character at the end

- Multiline String

```
'''
This course is awesome.
I'm going to start Project 1
right now.
'''
```

Variables are simply labels/refs like C++ pointers

Python Variable

```
x = 3
y = 5
a = 3
x = ‘python’
```

Python Variable

```
x = 3
y = 5
a = 3
x = ‘python’
```

Stack

Heap

Python Variable

```
x = 3
y = 5
a = 3
x = ‘python’
```

Stack

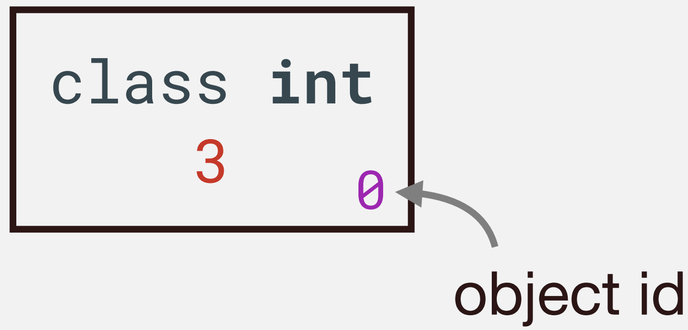
Heap

Python Variable

```
x = 3
y = 5
a = 3
x = ‘python’
```

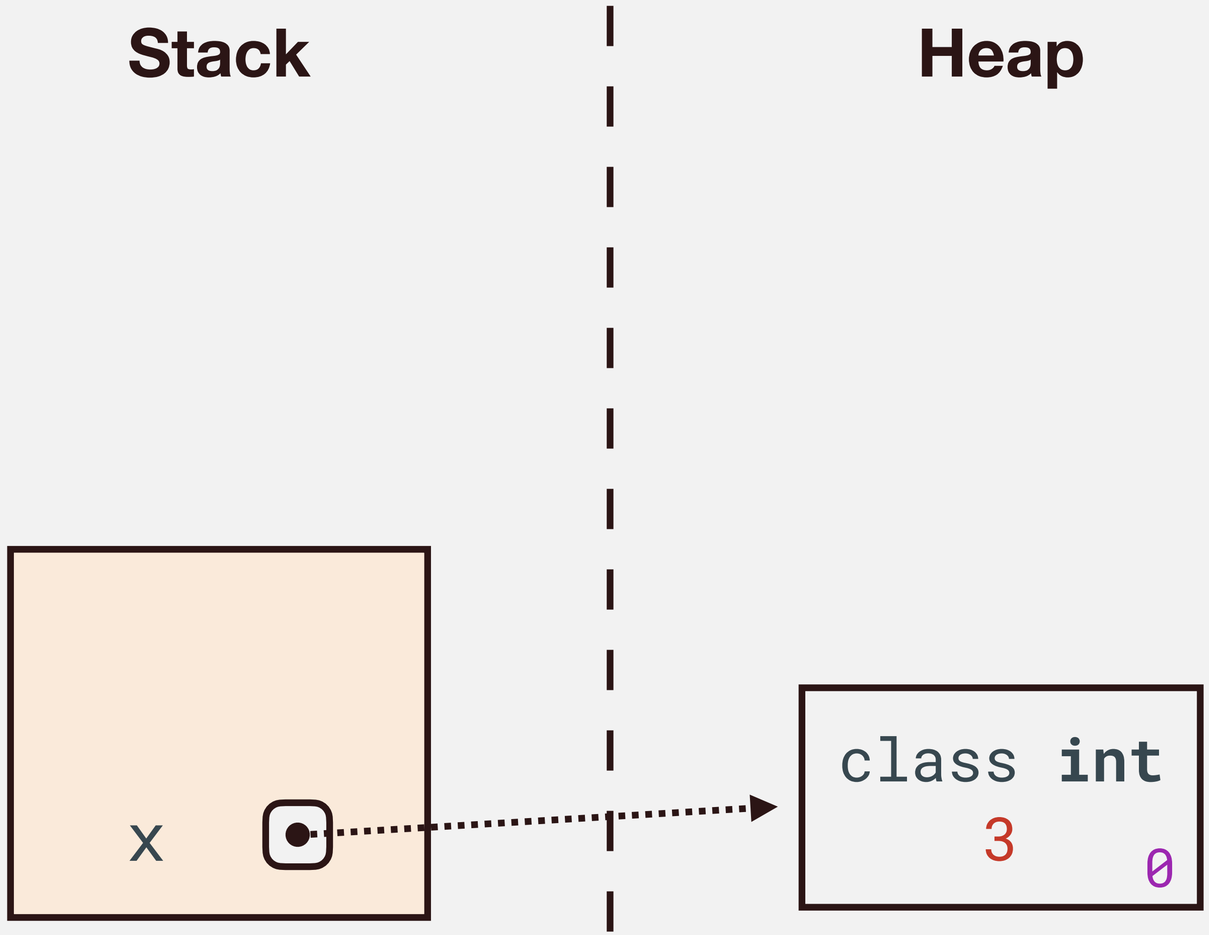
Stack

Heap



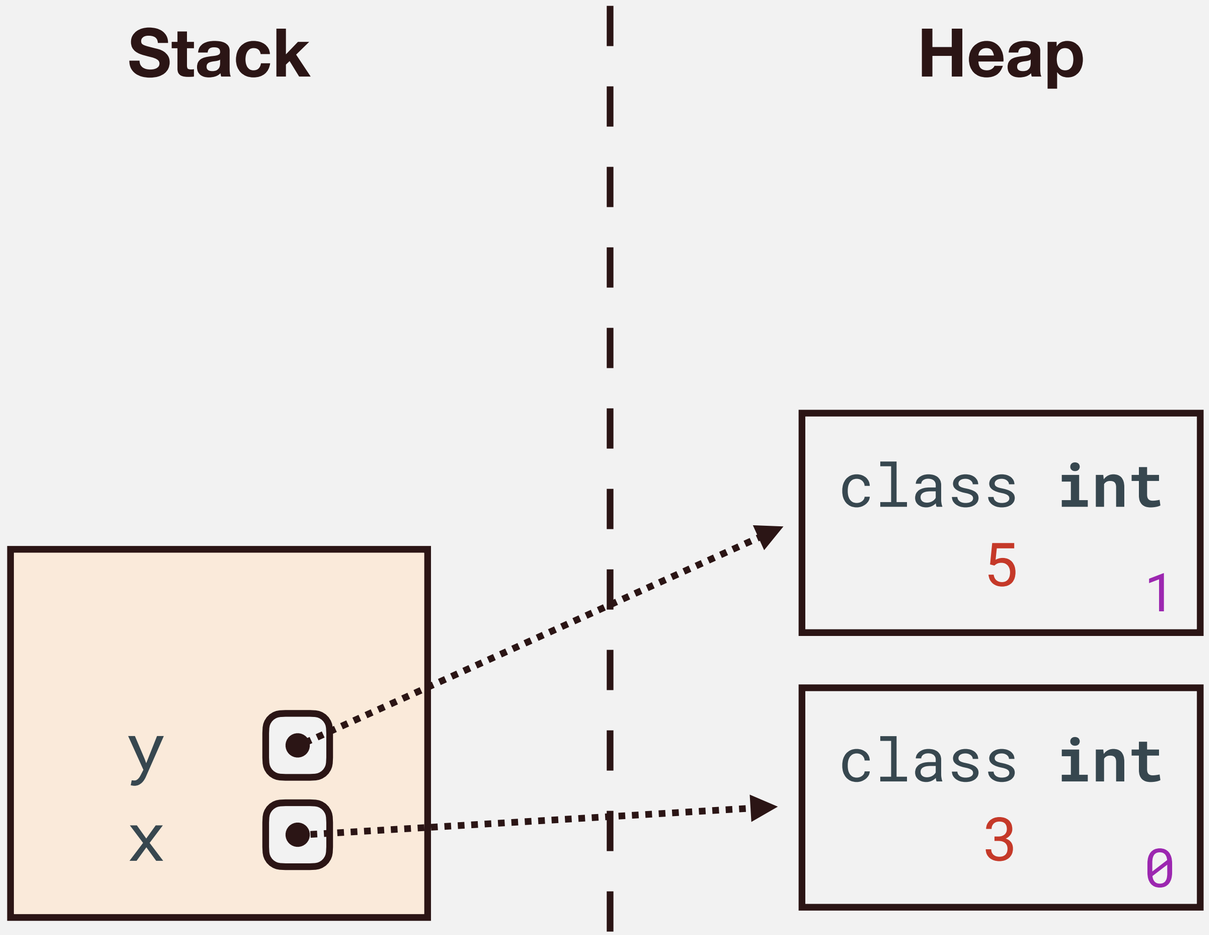
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
```



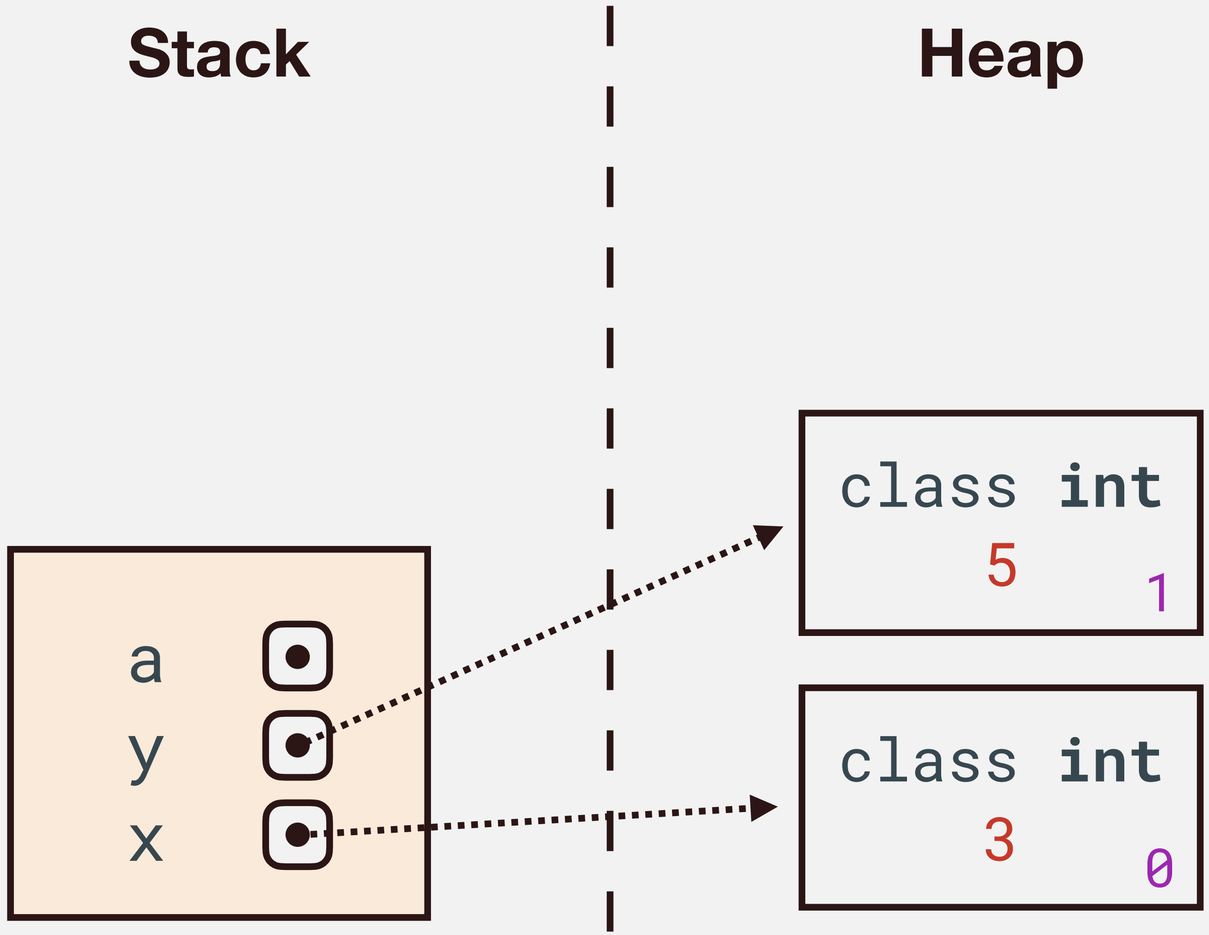
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
```



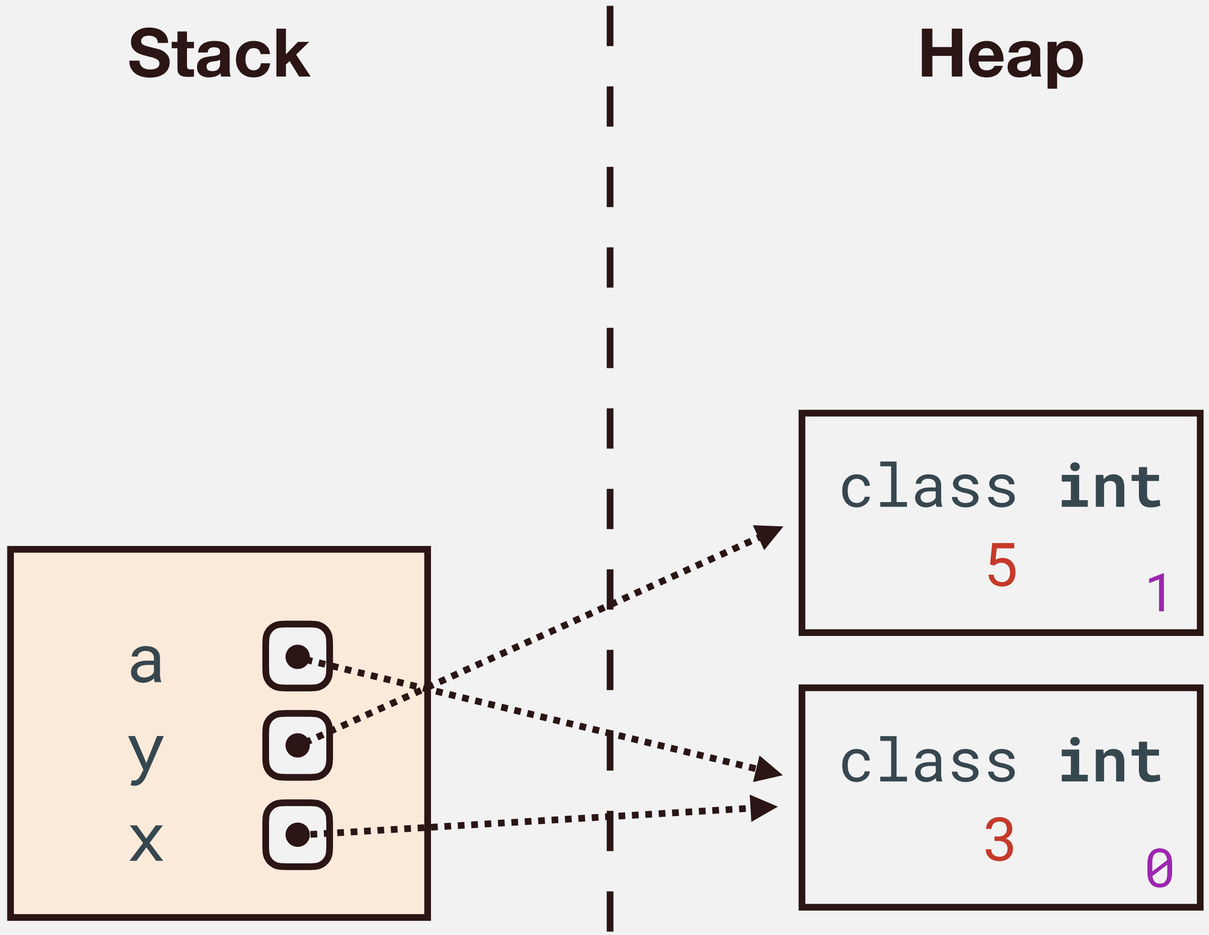
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
```



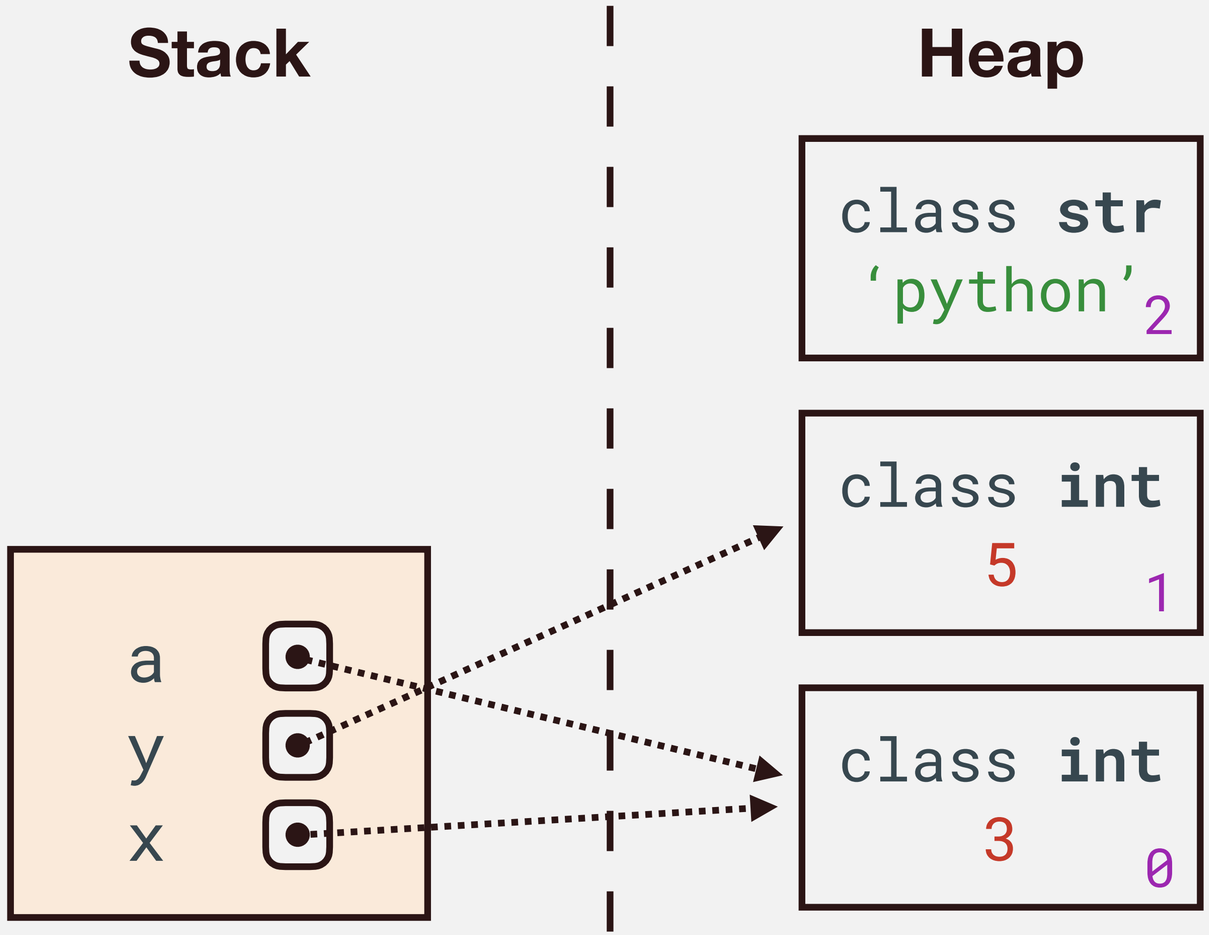
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
```



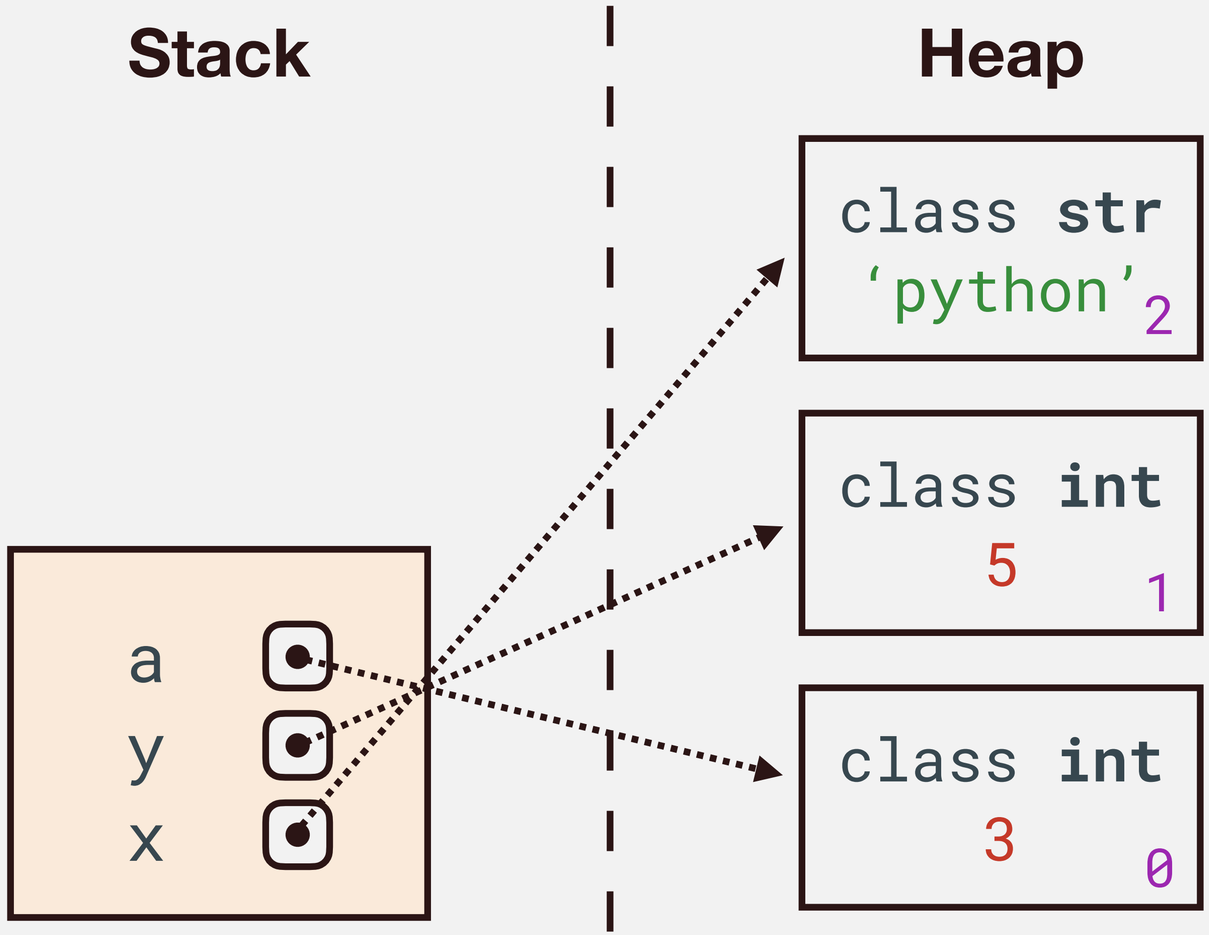
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
```



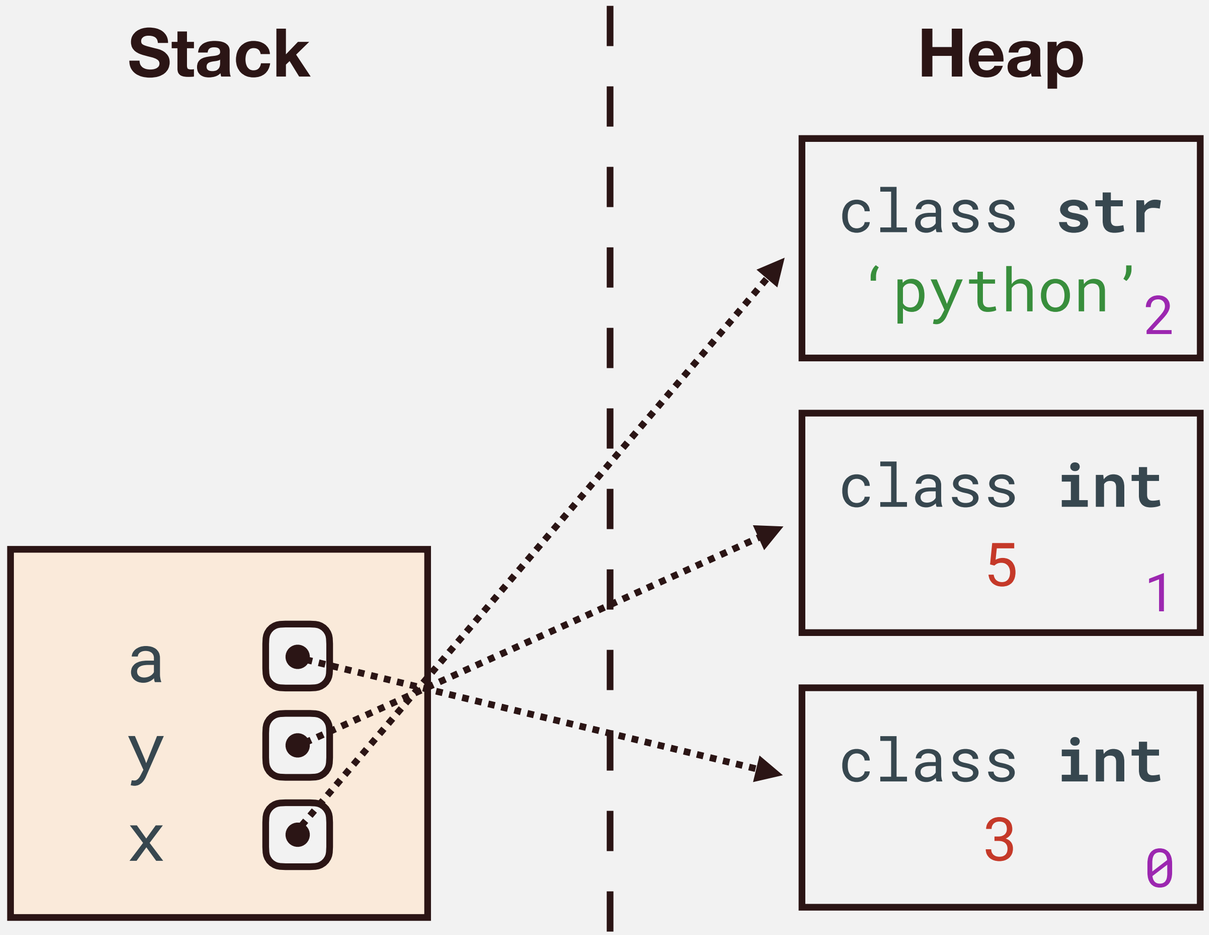
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
```



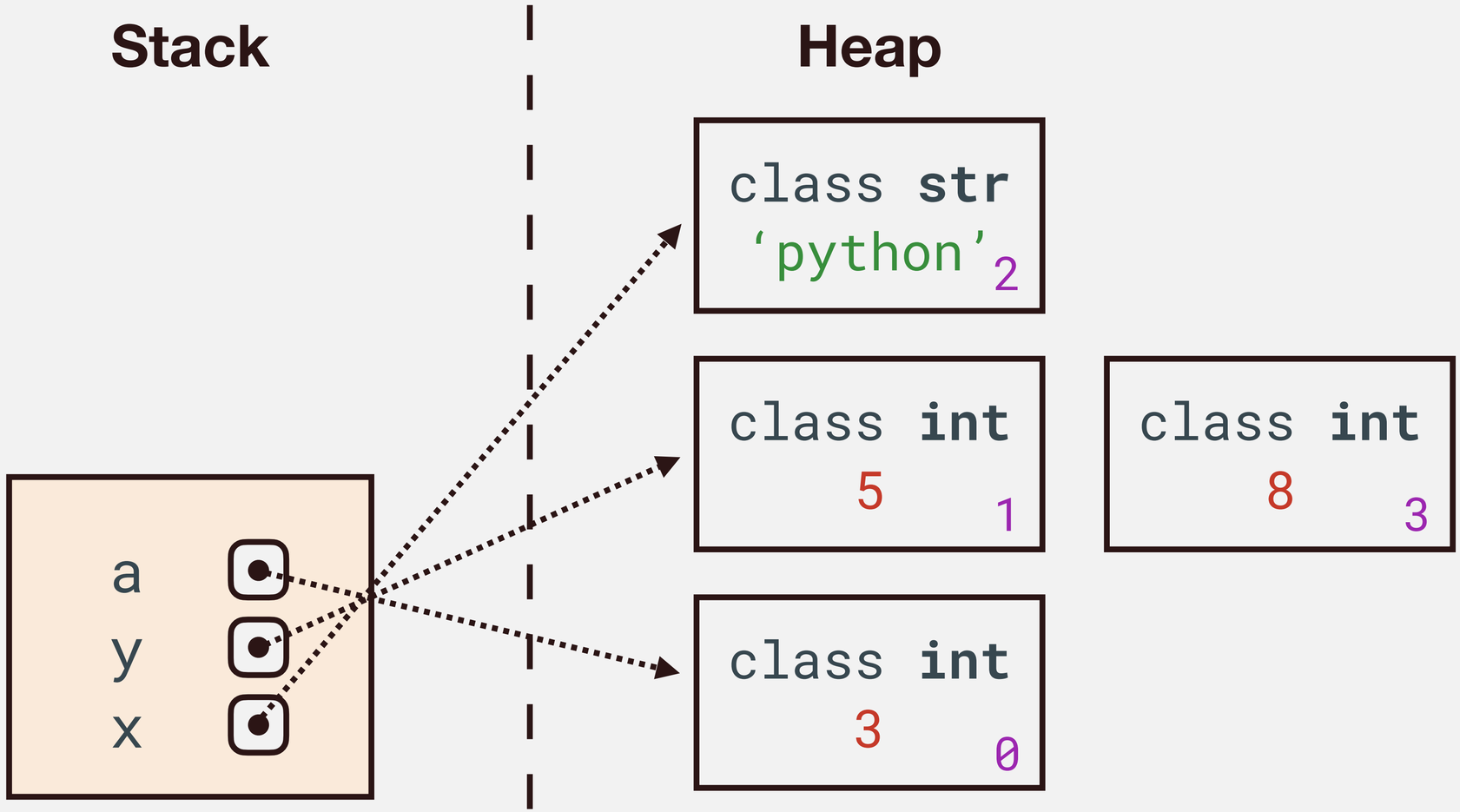
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
y = y + 3
```



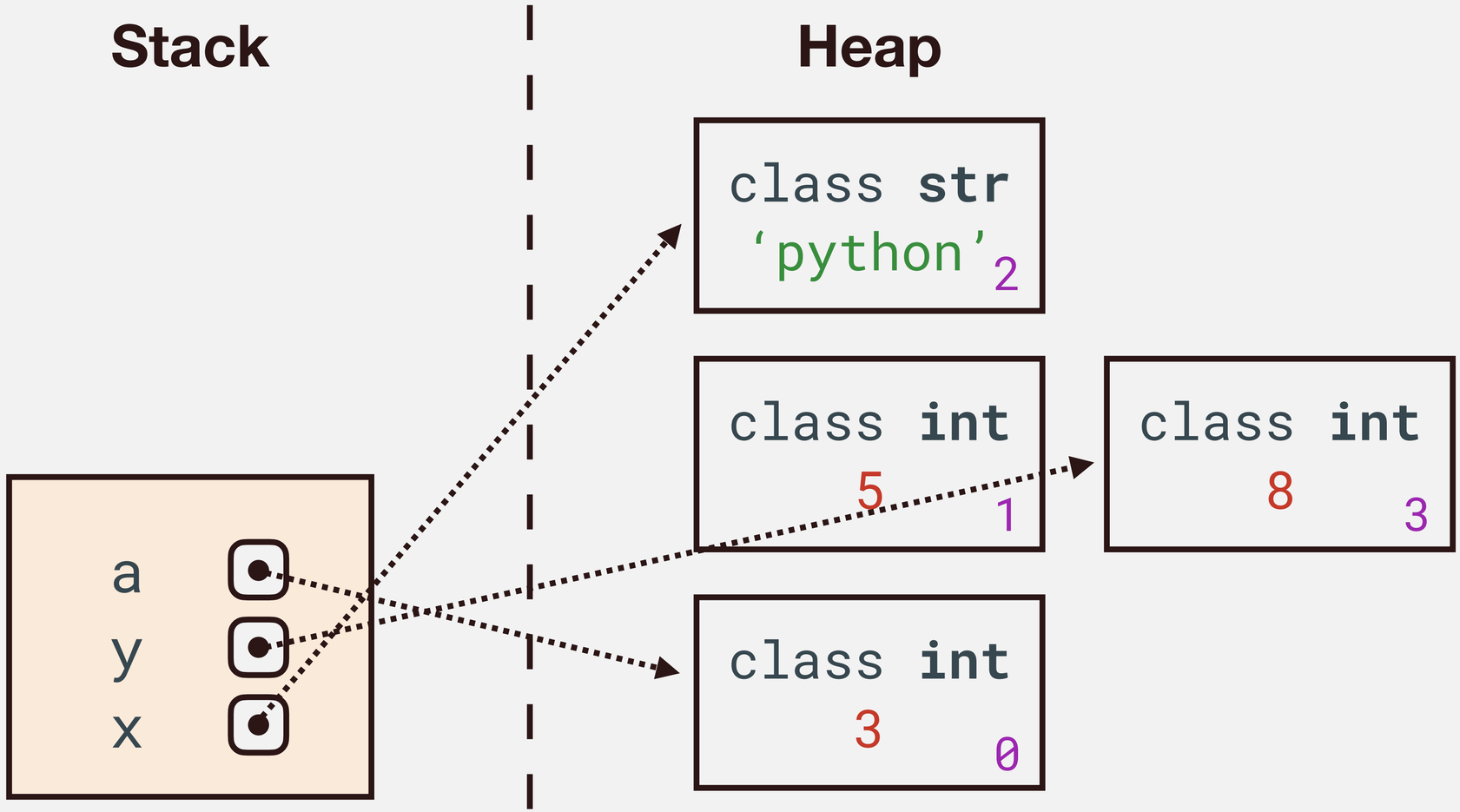
Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
y = y + 3
```



Python Variable

```
x = 3
y = 5
a = 3
x = 'python'
y = y + 3
```



Operators

	C++	Python
Assignment	=	=
Arithmetic	+ - * / %	+ - * / % ** //
Logical	&& || !	and or not
Bitwise	& | ^ << >> ~	& | ^ << >> ~
Relational	== != > >= < <=	== != > >= < <=
Short-cut	+= -= *= /= %= …	+= -= *= /= %= …

Operators

	C++	Python
Inc/Dec	`++ --`	✗
Conditional	`?:`	✗
Identity	✗	`is` `is not`
Membership	✗	`in` `not in`

Operators

	C++	Python
Inc/Dec	`++ --`	✗
Conditional	`?:`	✗
Identity	✗	`is` `is not`
Membership	✗	`in` `not in`

```
a, b = 1, 2;
a, b = b, a;
```

Sequence Types

List

➜ No fixed-sized array in Python. **List** is a better alternative!

List

- No fixed-sized array in Python. **List** is a better alternative!
- Initialize

```
a = [0, 1, 1, 2, 3, 5, 8]
a = [‘this’, ‘course’, ‘is’, ‘awesome’]
```

List

- No fixed-sized array in Python. **List** is a better alternative!
- Initialize

```
a = [0, 1, 1, 2, 3, 5, 8]
a = ['this', 'course', 'is', 'awesome']
a = []
```

List

- No fixed-sized array in Python. **List** is a better alternative!
- Initialize

```
a = [0, 1, 1, 2, 3, 5, 8]
a = ['this', 'course', 'is', 'awesome']
a = []
a = list('this')
```

List

- No fixed-sized array in Python. **List** is a better alternative!
- Initialize

```
a = [0, 1, 1, 2, 3, 5, 8]
a = ['this', 'course', 'is', 'awesome']
a = []
a = list('this')
a = [0, 1, [1, 2], [3, 5], 8]
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
x = a[2] # 1
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
x = a[2] # 1
y = a[-1] # 8
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
x = a[2] # 1
y = a[-1] # 8

a = [0, 1, [1, 2], [3, 5], 8]
l = len(a)
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
x = a[2] # 1
y = a[-1] # 8

a = [0, 1, [1, 2], [3, 5], 8]
l = len(a) # 5
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
x = a[2] # 1
y = a[-1] # 8

a = [0, 1, [1, 2], [3, 5], 8]
l = len(a) # 5
x = a[2] # [3, 5]
```

List Access

```
a = [0, 1, 1, 2, 3, 5, 8]
l = len(a) # 7
x = a[2] # 1
y = a[-1] # 8

a = [0, 1, [1, 2], [3, 5], 8]
l = len(a) # 5
x = a[2] # [3, 5]
y = a[2][1] # 5
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3] # [0, 1, 1, 2]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3] # [0, 1, 1, 2]
e = a[5:] # [5, 8]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3] # [0, 1, 1, 2]
e = a[5:] # [5, 8]
f = a[:] # [0, 1, 1, 2, 3, 5, 8]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3] # [0, 1, 1, 2]
e = a[5:] # [5, 8]
f = a[:] # [0, 1, 1, 2, 3, 5, 8]
g = a[0:5:2] # [0, 1, 3]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3] # [0, 1, 1, 2]
e = a[5:] # [5, 8]
f = a[:] # [0, 1, 1, 2, 3, 5, 8]
g = a[0:5:2] # [0, 1, 3]
h = a[::-1]
```

List Slicing

```
a = [0, 1, 1, 2, 3, 5, 8]
b = a[1:5] # [1, 1, 2, 3]
c = a[:3] # [0, 1, 1]
d = a[:-3] # [0, 1, 1, 2]
e = a[5:] # [5, 8]
f = a[:] # [0, 1, 1, 2, 3, 5, 8]
g = a[0:5:2] # [0, 1, 3]
h = a[::-1] # [8, 5, 3, 2, 1, 1, 0]
```

List Modification

```
a = [0, 1, 1, 2, 3, 5, 8]
a.append(13) # [0, 1, 1, 2, 3, 5, 8, 13]
```

List Modification

```
a = [0, 1, 1, 2, 3, 5, 8]
a.append(13) # [0, 1, 1, 2, 3, 5, 8, 13]
a.insert(5, 1) # [0, 1, 1, 2, 3, 1, 5, 8, 13]
```

List Modification

```
a = [0, 1, 1, 2, 3, 5, 8]
a.append(13) # [0, 1, 1, 2, 3, 5, 8, 13]
a.insert(5, 1) # [0, 1, 1, 2, 3, 1, 5, 8, 13]
a[5] = 2 # [0, 1, 1, 2, 3, 2, 5, 8, 13]
```

List Modification

```
a = [0, 1, 1, 2, 3, 5, 8]
a.append(13) # [0, 1, 1, 2, 3, 5, 8, 13]
a.insert(5, 1) # [0, 1, 1, 2, 3, 1, 5, 8, 13]
a[5] = 2 # [0, 1, 1, 2, 3, 2, 5, 8, 13]
a.reverse() # [13, 8, 5, 2, 3, 2, 1, 1, 0]
a.sort() # [0, 1, 1, 2, 2, 3, 5, 8, 13]
a.remove(2) # [0, 1, 1, 2, 3, 5, 8, 13]
```

List Modification

```
a = [0, 1, 1, 2, 3, 5, 8]
a.append(13) # [0, 1, 1, 2, 3, 5, 8, 13]
a.insert(5, 1) # [0, 1, 1, 2, 3, 1, 5, 8, 13]
a[5] = 2 # [0, 1, 1, 2, 3, 2, 5, 8, 13]
a.reverse() # [13, 8, 5, 2, 3, 2, 1, 1, 0]
a.sort() # [0, 1, 1, 2, 2, 3, 5, 8, 13]
a.remove(2) # [0, 1, 1, 2, 3, 5, 8, 13]
```

Refer to: docs.python.org

List Operators

```
a = [0, 1] + [1, 2, 3] # [0, 1, 1, 2, 3]
```

List Operators

```
a = [0, 1] + [1, 2, 3] # [0, 1, 1, 2, 3]
a = [0, 1] * 3 # [0, 1, 0, 1, 0, 1]
```

List Operators

```
a = [0, 1] + [1, 2, 3] # [0, 1, 1, 2, 3]
a = [0, 1] * 3 # [0, 1, 0, 1, 0, 1]
del a[2] # [0, 1, 1, 0, 1]
```

List Operators

```
a = [0, 1] + [1, 2, 3] # [0, 1, 1, 2, 3]
a = [0, 1] * 3 # [0, 1, 0, 1, 0, 1]
del a[2] # [0, 1, 1, 0, 1]
x = 1 in [1, 2, 3] # True
x = 2 not in [1, 2, 3] # False
```

Tuple

➜ **Immutable** sequence

```
vec = (1, 3, 5)
```

Tuple

➜ **Immutable** sequence

```
vec = (1, 3, 5)
x = vec[1] # 3
y = vec[:-1] # (1, 3)
```

Tuple

➜ **Immutable** sequence

```
vec = (1, 3, 5)
x = vec[1] # 3
y = vec[:-1] # (1, 3)
vec[1] = 7  ❌
vec.append(7) ❌
```

Tuple

➜ **Immutable** sequence

```
vec = (1, 3, 5)
x = vec[1] # 3
y = vec[:-1] # (1, 3)
vec[1] = 7  ❌
vec.append(7) ❌
vec = (1, 3, 5) + (7, 9) # (1, 3, 5, 7, 9)
```

Tuple

➜ **Immutable** sequence

```
vec = (1, 3, 5)
x = vec[1] # 3
y = vec[:-1] # (1, 3)
vec[1] = 7 ❌
vec.append(7) ❌
vec = (1, 3, 5) + (7, 9) # (1, 3, 5, 7, 9)
a = list((1, 3, 5)) # [1, 3, 5]
vec = tuple([1, 3, 5]) # (1, 3, 5)
```

Range

➜ **Immutable** sequence of numbers

```
r = range(0,10,2) # 0, 2, 4, 6, 8
r = range(1,10)
```

Range

➜ **Immutable** sequence of numbers

```
r = range(0,10,2) # 0, 2, 4, 6, 8
r = range(1,10)
x = r[5] # 6
```

Range

➜ **Immutable** sequence of numbers

```
r = range(0,10,2) # 0, 2, 4, 6, 8
r = range(1,10)
x = r[5] # 6
y = r[:5]
```

Range

➜ **Immutable** sequence of numbers

```
r = range(0,10,2) # 0, 2, 4, 6, 8
r = range(1,10)
x = r[5] # 6
y = r[:5] # range(1, 6)
```

Range

➜ **Immutable** sequence of numbers

```
r = range(0,10,2) # 0, 2, 4, 6, 8
r = range(1,10)
x = r[5] # 6
y = r[:5] # range(1, 6)
s = 0
for i in range(1, 10):
  s += i
```

String

➜ **Immutable** text sequence

```
s = ‘python’
a = s[-1] # ’n’
b = s[2:] # ‘thon’
c = s * 2 # ‘pythonpython’
```

String

➜ **Immutable** text sequence

```
s = ‘python’
a = s[-1] # ’n’
b = s[2:] # ‘thon’
c = s * 2 # ‘pythonpython’
if ‘n’ in s:
  print(‘whatever’)
```

String

➜ **Immutable** text sequence

```
s = ‘python’
a = s[-1] # ’n’
b = s[2:] # ‘thon’
c = s * 2 # ‘pythonpython’
if ‘n’ in s:
  print(‘whatever’)
s = ‘Celtics, Lakers, Warriors, Mavericks’
```

String

➜ **Immutable** text sequence

```
s = ‘python’
a = s[-1] # ’n’
b = s[2:] # ‘thon’
c = s * 2 # ‘pythonpython’
if ‘n’ in s:
  print(‘whatever’)
s = ‘Celtics, Lakers, Warriors, Mavericks’
a = s.split(‘,’) # [‘Celtics’, ‘Lakers’, ‘Warriors’, ‘Mavericks’]
```

String

➜ **Immutable** text sequence

```
s = ‘python’
a = s[-1] # ’n’
b = s[2:] # ‘thon’
c = s * 2 # ‘pythonpython’
if ‘n’ in s:
  print(‘whatever’)
s = ‘Celtics, Lakers, Warriors, Mavericks’
a = s.split(‘,’) # [‘Celtics’, ‘Lakers’, ‘Warriors’, ‘Mavericks’]
```

Refer to: docs.python.org

Set

- Similar to C++ STL unordered_set
- Useful for membership testing and removing duplicates from a sequence

```
s = {1, 4, 9, 16, 25, 36}
```

Set

- Similar to C++ STL unordered_set
- Useful for membership testing and removing duplicates from a sequence

```
s = {1, 4, 9, 16, 25, 36}
n = int(input('Please enter a number: '))
if n in s:
  print(f'{n} is a perfect square.')
```

Set

- Similar to C++ STL unordered_set
- Useful for membership testing and removing duplicates from a sequence

```
s = {1, 4, 9, 16, 25, 36}
n = int(input('Please enter a number: '))
if n in s:
  print(f'{n} is a perfect square.')
```

Much faster than List

Set

- Similar to C++ STL unordered_set
- Useful for membership testing and removing duplicates from a sequence

```
s = {1, 4, 9, 16, 25, 36}
n = int(input('Please enter a number: '))
if n in s: ⟵ Much faster than List
  print(f'{n} is a perfect square.')
a = ['we', 'will', 'we', 'will', 'rock', 'you']
s = set(a)
a = list(s) # ['will', 'we', 'rock', 'you']
```

Set Operations

Math	Python	
$A \cap B$	`A & B`	`A.intersection(B)`
$A \cup B$	`A | B`	`A.union(B)`
$A \setminus B$	`A - B`	`A.difference(B)`
$x \in A$	`x in A`	
$A \subset B$	`A < B`	
$A \subseteq B$	`A <= B`	`A.issubset(B)`

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {‘name’:’Stephan Curry’, ‘age’:34, ‘championship’:4}
```

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}


d = {[1, 2]:2, [3, 4, 5]:3}
```

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}
# keys must be hashable (e.g., immutable built-in objects)
d = {[1, 2]:2, [3, 4, 5]:3}
```

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}
# keys must be hashable (e.g., immutable built-in objects)
d = {[1, 2]:2, [3, 4, 5]:3} ❌
players = {'James':206, 'Curry':188, 'Tatum':203}
x = players['Curry'] # 188
```

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}
# keys must be hashable (e.g., immutable built-in objects)
d = {[1, 2]:2, [3, 4, 5]:3} ❌
players = {'James':206, 'Curry':188, 'Tatum':203}
x = players['Curry'] # 188
y = players['Doncic']
```

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}
# keys must be hashable (e.g., immutable built-in objects)
d = {[1, 2]:2, [3, 4, 5]:3} ❌
players = {'James':206, 'Curry':188, 'Tatum':203}
x = players['Curry'] # 188
y = players['Doncic'] # KeyError! ❌
```

Dictionary

➜ Similar to C++ STL unordered_map

```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}
# keys must be hashable (e.g., immutable built-in objects)
d = {[1, 2]:2, [3, 4, 5]:3} ❌
players = {'James':206, 'Curry':188, 'Tatum':203}
x = players['Curry'] # 188
y = players['Doncic'] # KeyError! ❌
players['Doncic'] = 201
```

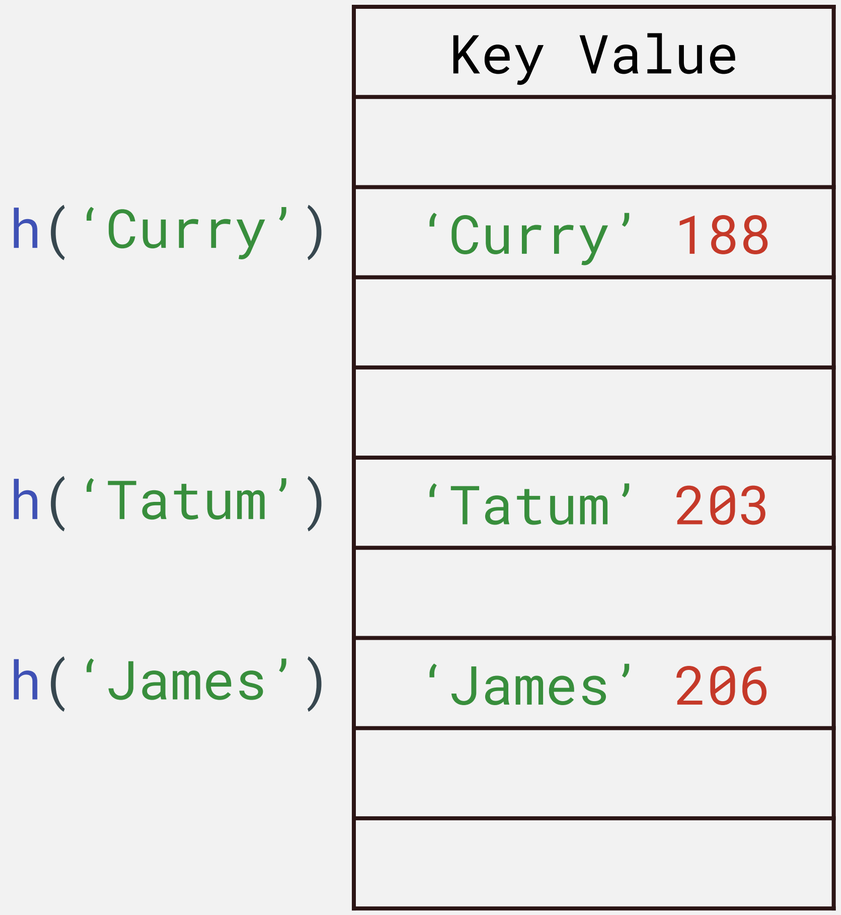
Dictionary

➜ Similar to C++ STL unordered_map

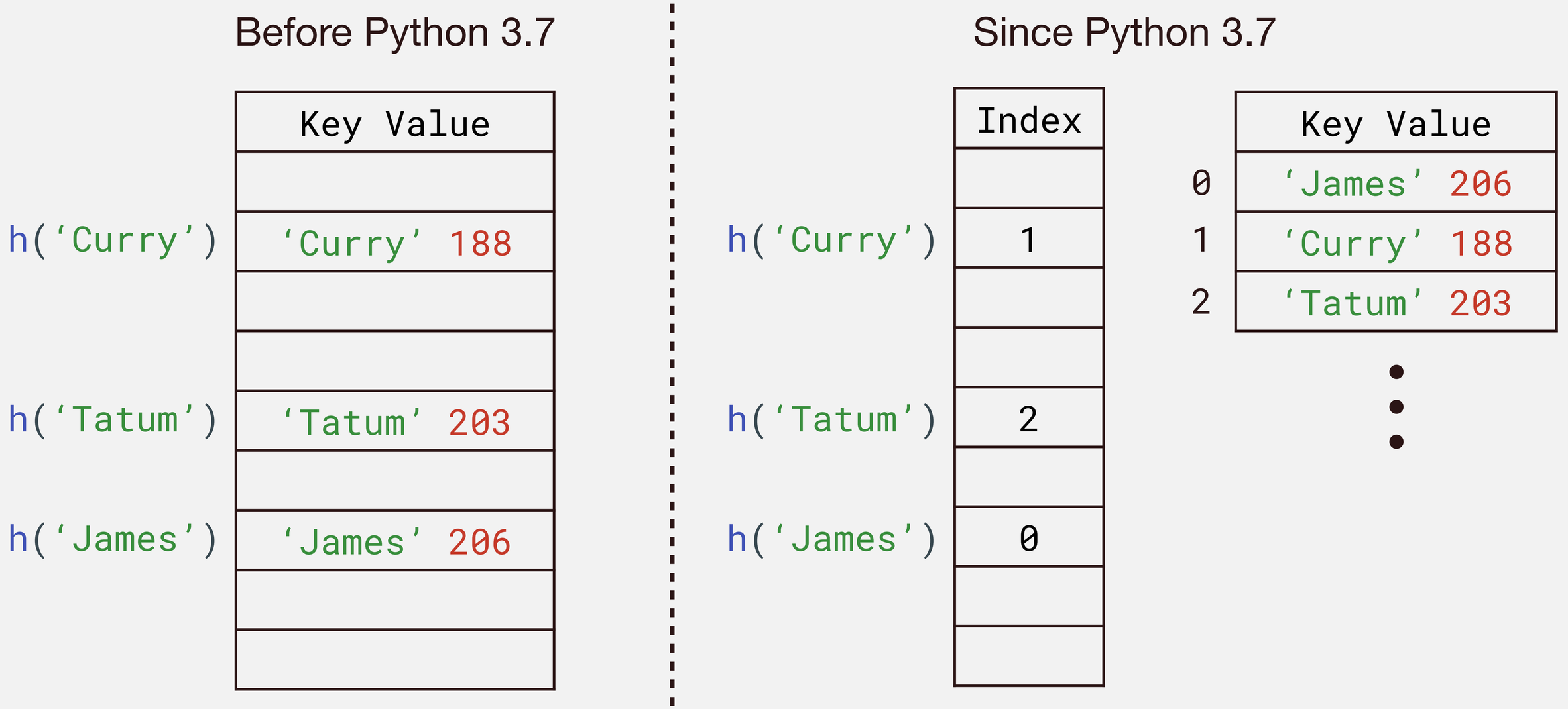
```
d = {}
d = {'name':'Stephan Curry', 'age':34, 'championship':4}
# keys must be hashable (e.g., immutable built-in objects)
d = {[1, 2]:2, [3, 4, 5]:3} ❌
players = {'James':206, 'Curry':188, 'Tatum':203}
x = players['Curry'] # 188
y = players['Doncic'] # KeyError! ❌
players['Doncic'] = 201
print(players) # {'James':206,'Curry':188,'Tatum':203,'Doncic':201}
```

Dictionary Keeps Insertion Order

Before Python 3.7



Dictionary Keeps Insertion Order



Control Follow

if-elif-else

C++

```cpp
if (x > 0) {
  std::cout << “positive\n”;
} else if (x == 0) {
  std::cout << “zero\n”;
} else {
  std::cout << “negative\n”;
}
```

Python

```python
if x > 0:
  print(“positive”)
elif x == 0:
  print(“zero”)
else:
  print(“negative”)
```

while

C++

```cpp
int i = 0;
while (i < 10) {
  std::cout << “汪\n”;
  i++;
}
```

Python

```python
i = 0
while (i < 10):
  print(“汪”)
  i += 1
```

for

C++

```cpp
int sum = 0;
for (int i = 0; i < 10; i++) {
  sum += i;
}
```

Python

```python
sum = 0
for i in range(10):
  sum += i
```

for

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
for word in sentence:
  print(word)
```

for

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
for word in sentence:
  print(word)

ppl_dict = {‘Beijing’:2189, ‘Guangdong’:12684, ‘Shanghai’:2489}
for city in ppl_dict:
  print(city)
```

for

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
for word in sentence:
  print(word)

ppl_dict = {‘Beijing’:2189, ‘Guangdong’:12684, ‘Shanghai’:2489}
for city in ppl_dict:
  print(city)
sum = 0
for city, ppl in ppl_dict.items():
  sum += ppl
```

List Comprehension

➜ Create a new list based on the values in an existing list

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
wordlen = []
for word in sentence:
  wordlen.append(len(word))
```

List Comprehension

➜ Create a new list based on the values in an existing list

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
wordlen = [len(word) for word in sentence]
```

List Comprehension

➜ Create a new list based on the values in an existing list

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
short_words = []
for word in sentence:
  if len(word) < 5:
    short_words.append(word)
```

List Comprehension

➜ Create a new list based on the values in an existing list

```
sentence = [‘this’, ‘course’, ‘is’, ‘awesome’]
short_words = [word for word in sentence if len(word) < 5]
```

Dictionary Comprehension

➜ Create a new dictionary based on the values in an existing dictionary

```
players = {‘James’:206, ‘Curry’:188, ‘Tatum’:203}
tall_players = {k: v for k, v in players.items() if v > 200}
```

zip

```
x = [1, 2, 3, 4]
y = [1.2, 2.4, 3.6, 4.8]
z = zip(x, y)
print(list(z)) # [(1, 1.2), (2, 2.4), (3, 3.6), (4, 4.8)]
```

Functions

Functions

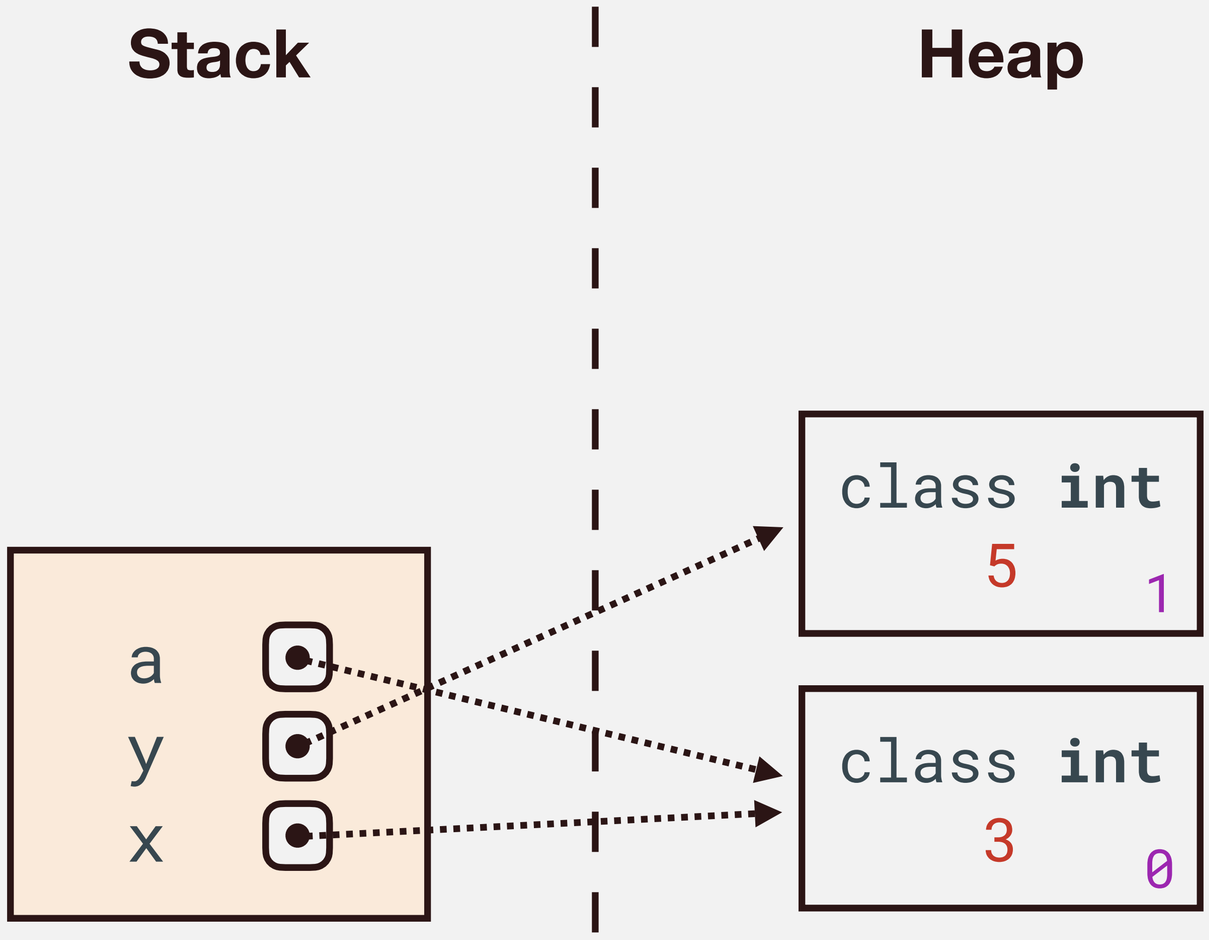
```
def picktall(players):
  players = {k: v for k, v in players.items() if v > 200}
  return len(players)
```

Pass by Object Reference

(Object Reference is passed by value)

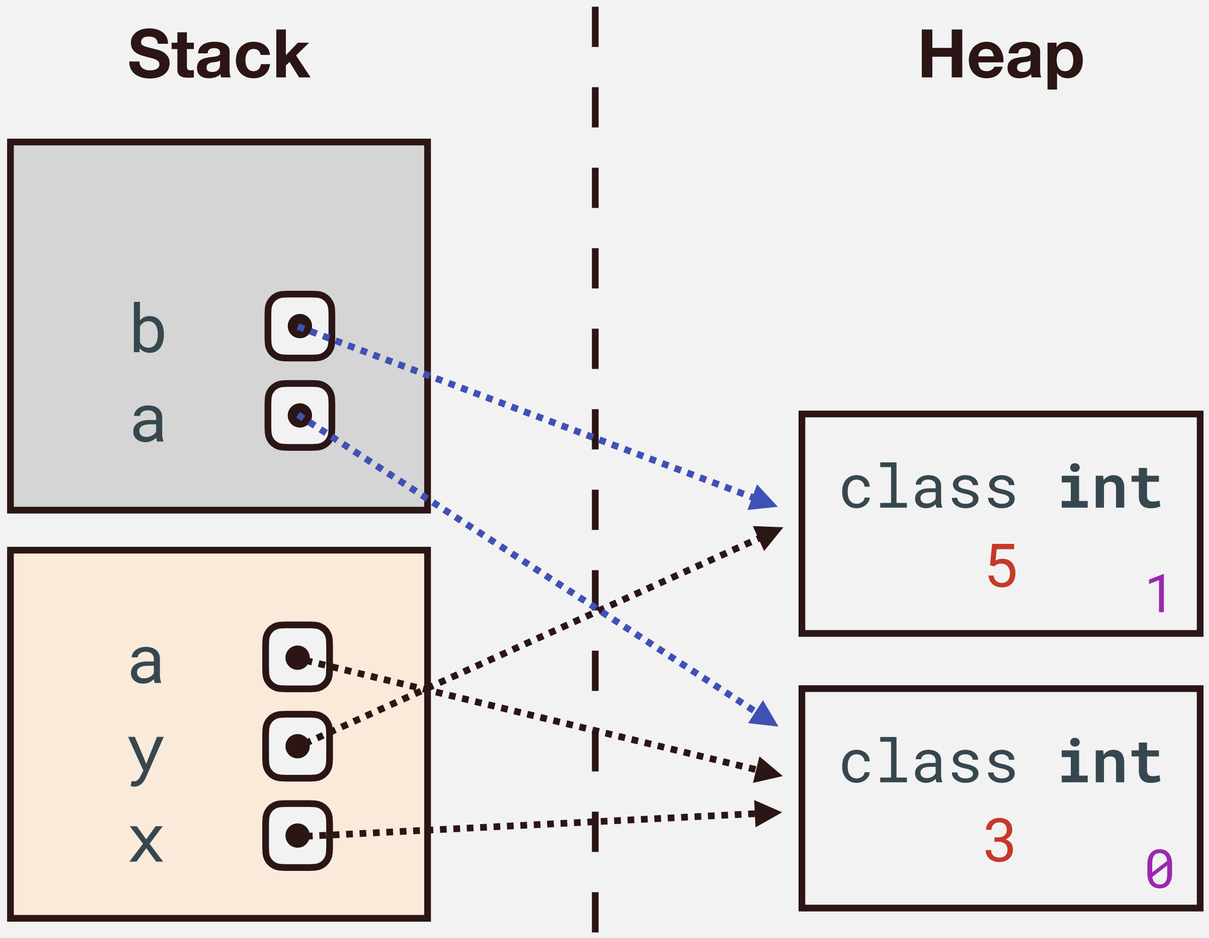
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c

z = func(x, y)
```



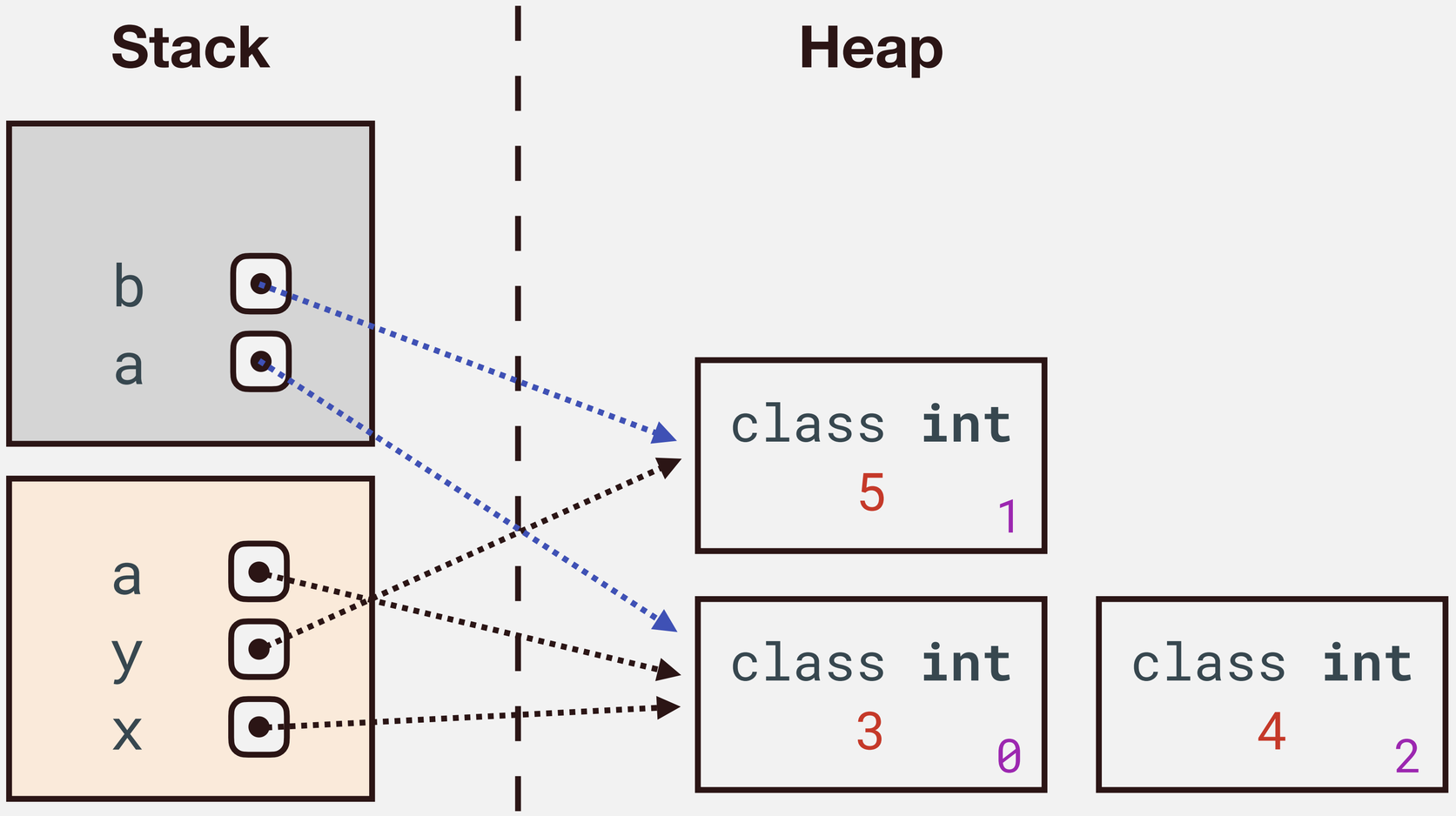
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c

z = func(x, y)
```



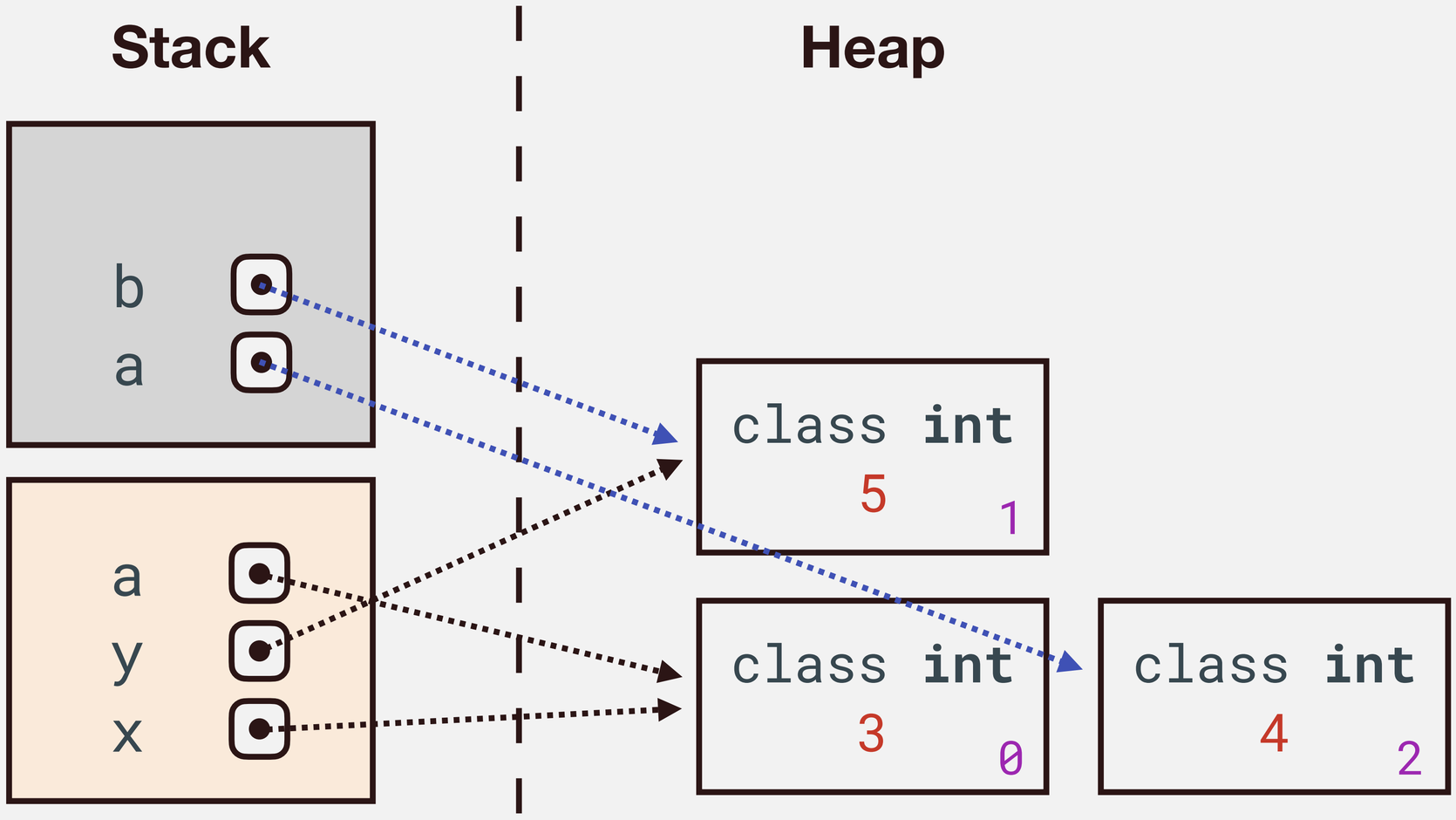
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c

z = func(x, y)
```



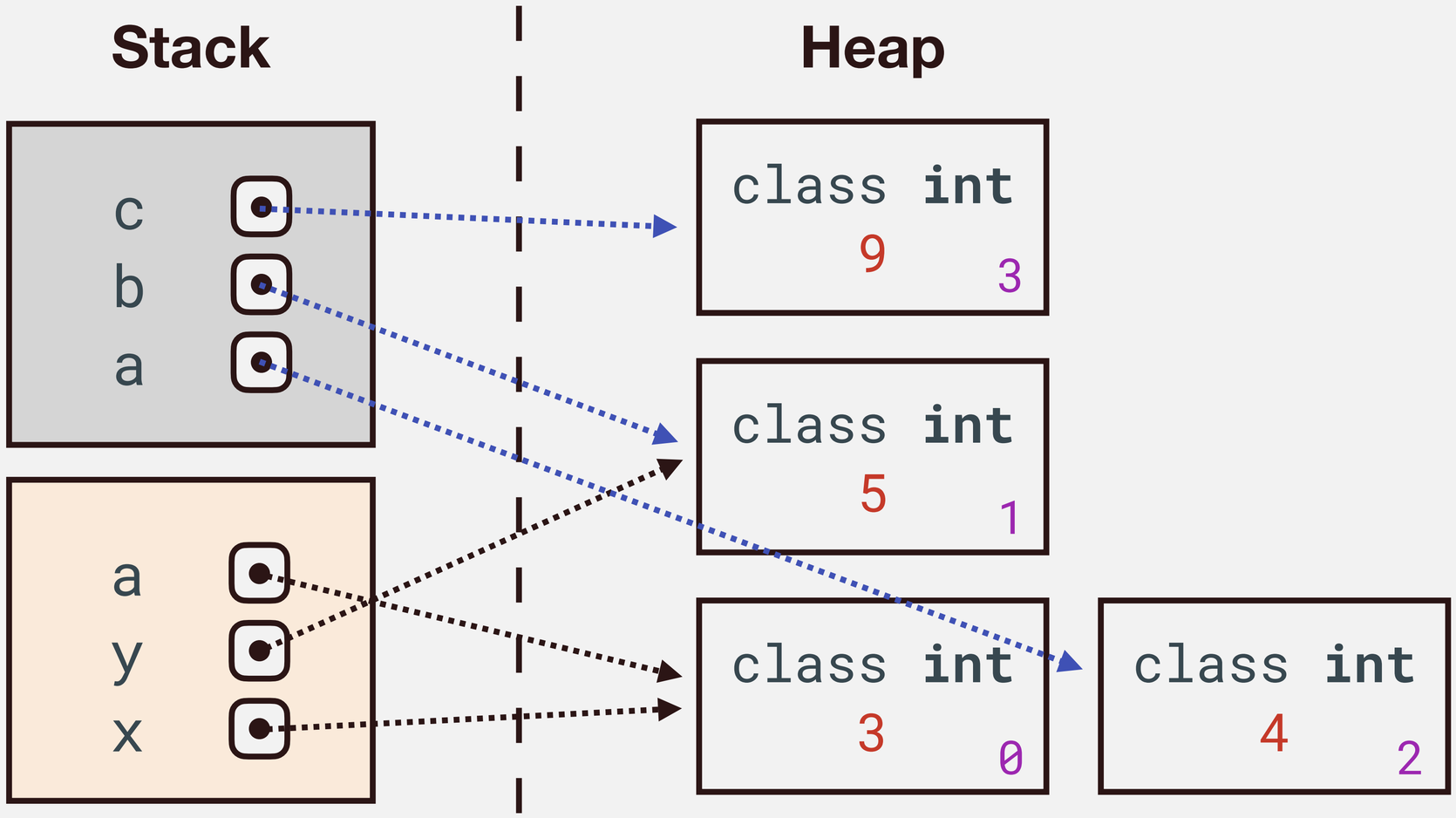
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c

z = func(x, y)
```



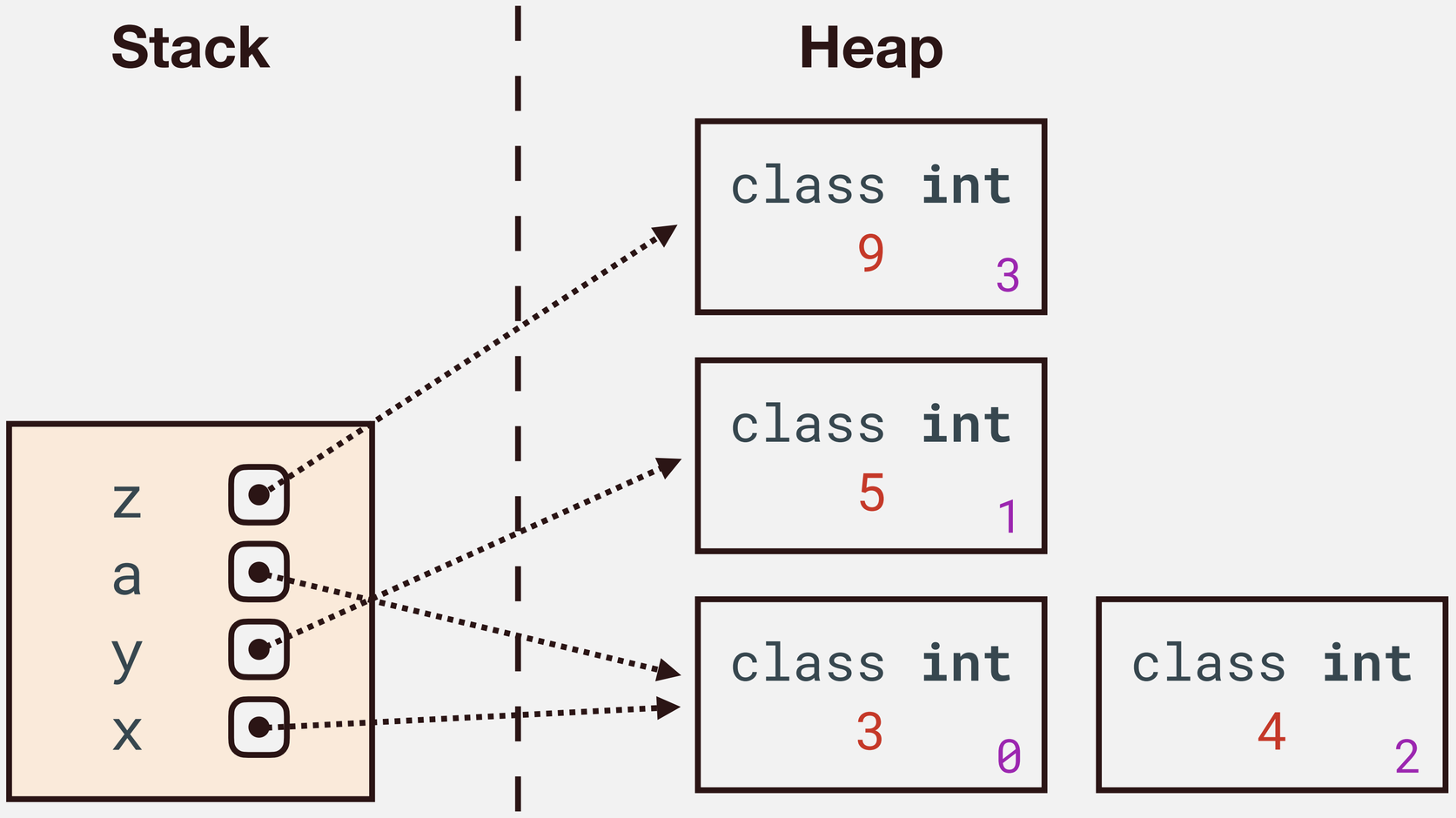
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c

z = func(x, y)
```



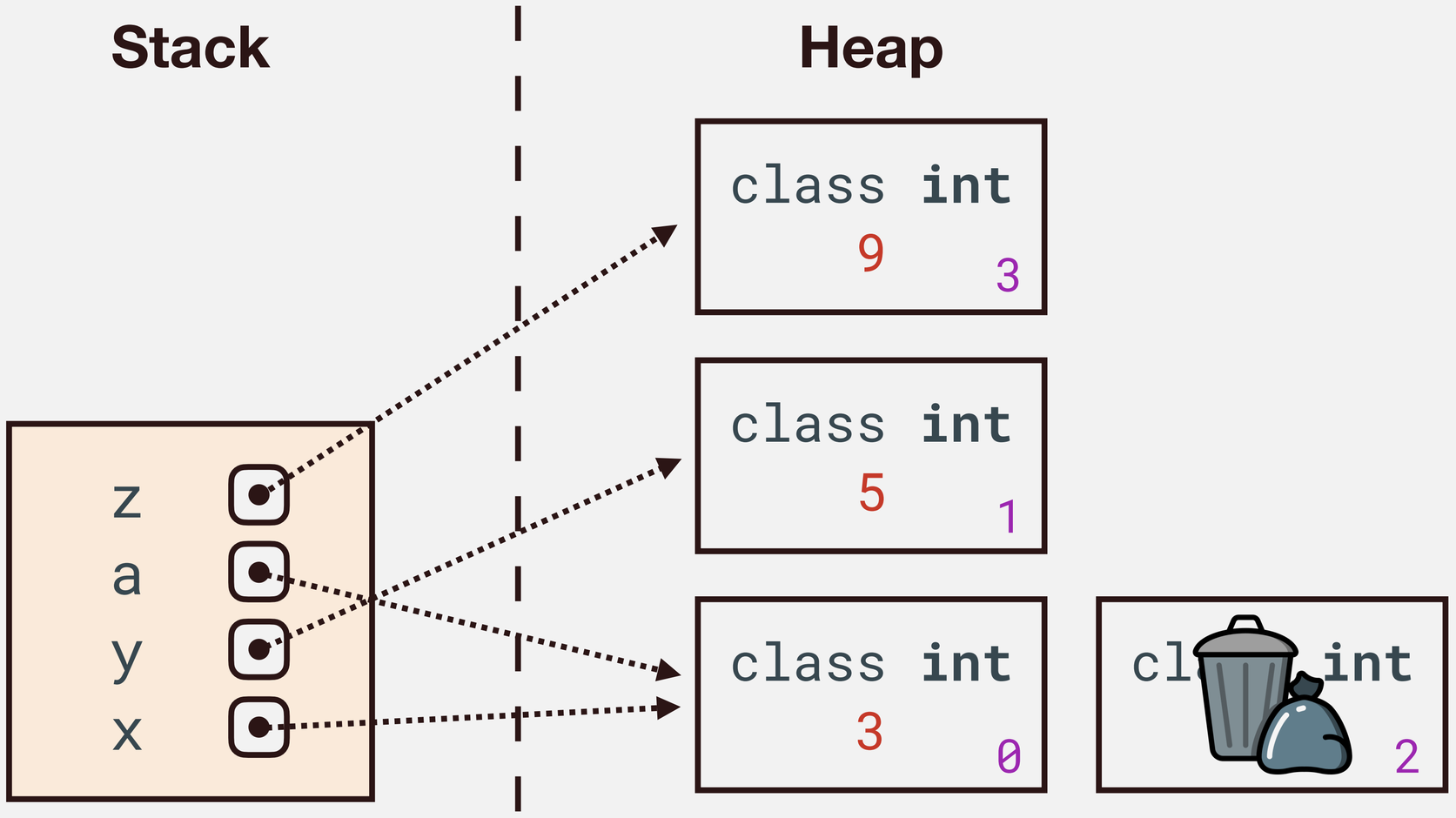
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c

z = func(x, y)
```



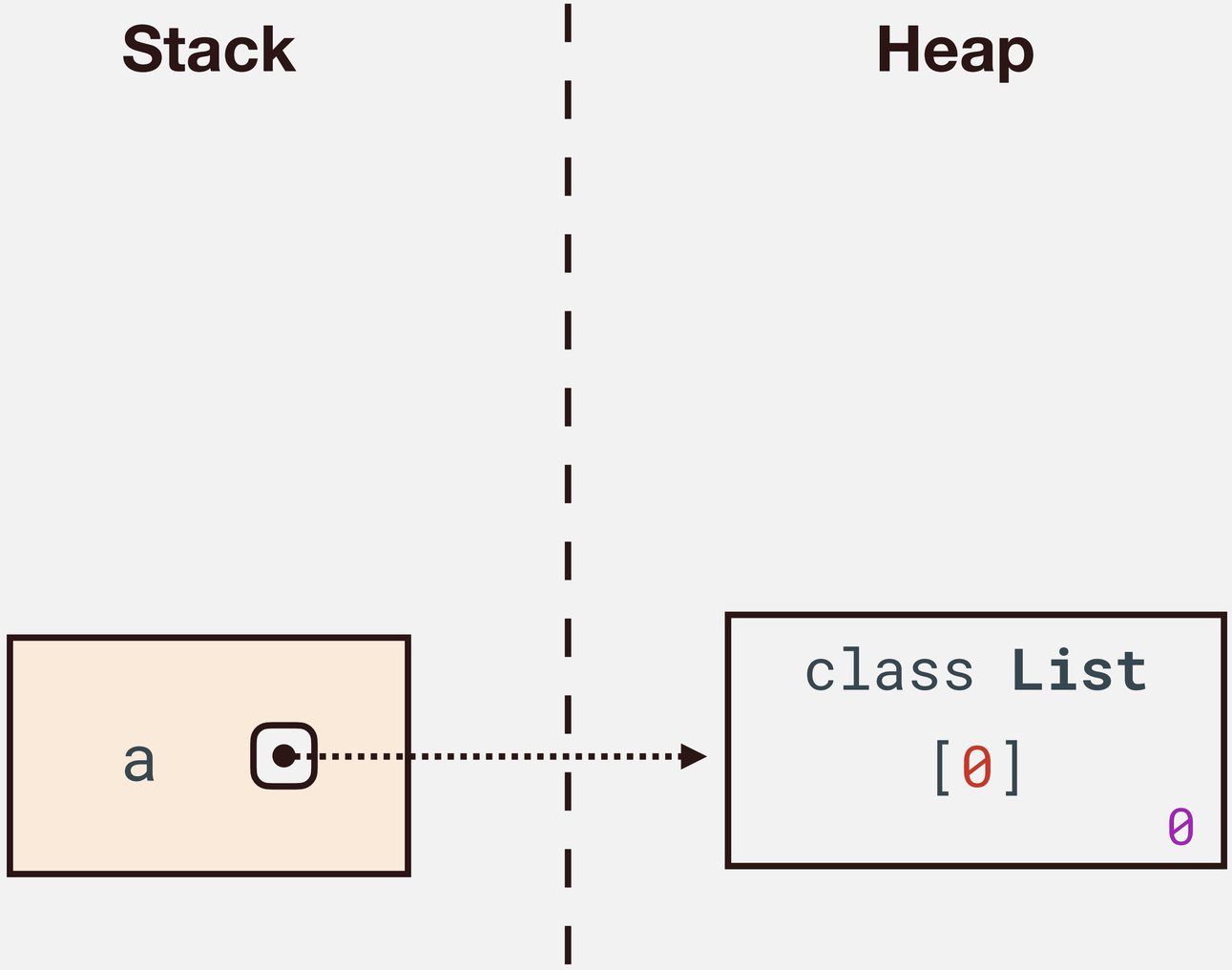
Pass by Object Reference

```
x = 3
y = 5
a = 3

def func(a, b):
    a += 1
    c = a + b
    return c


z = func(x, y)
```



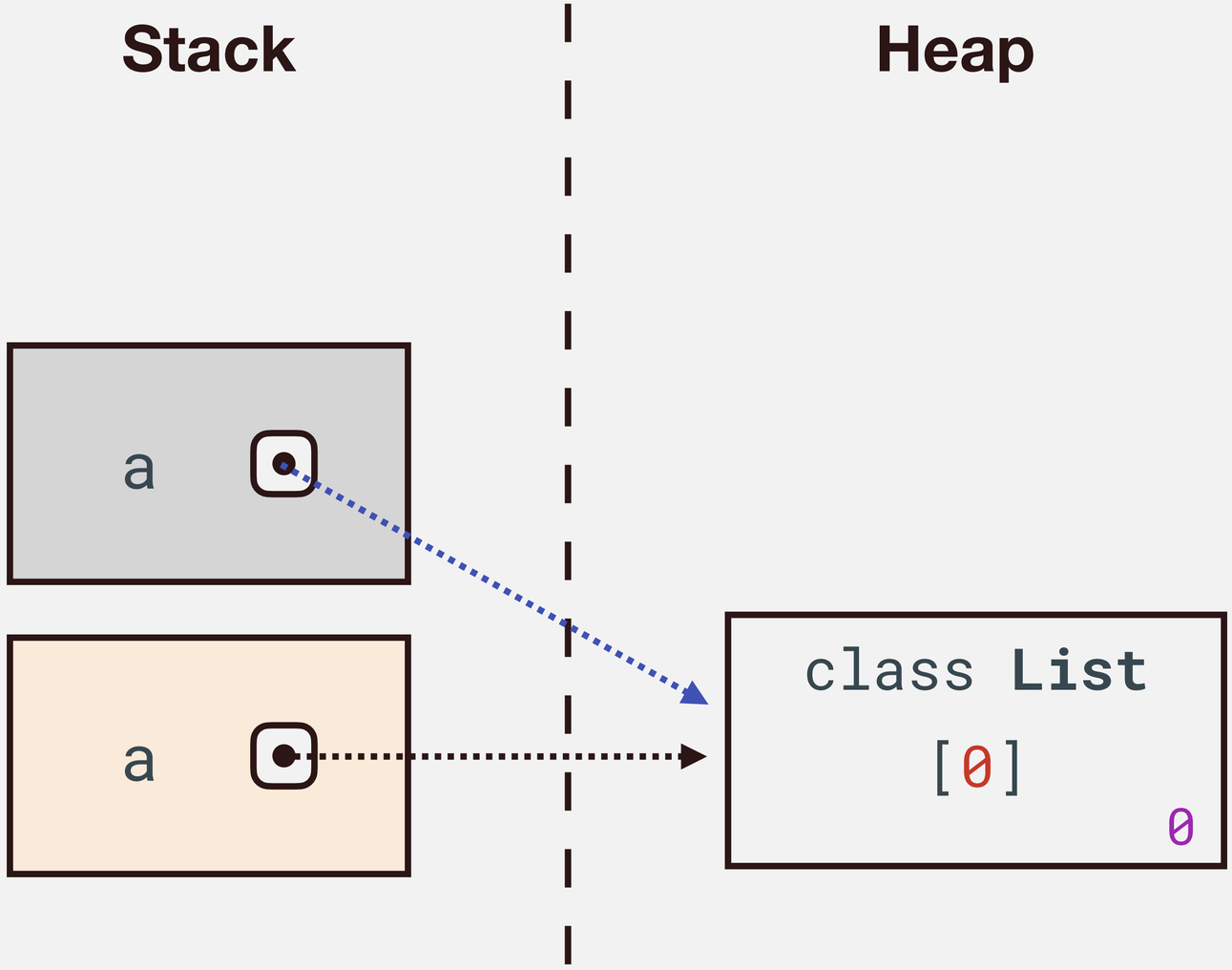
Pass by Object Reference

```
def reassign(a):
  a = [0, 1]

def append1(a):
  a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



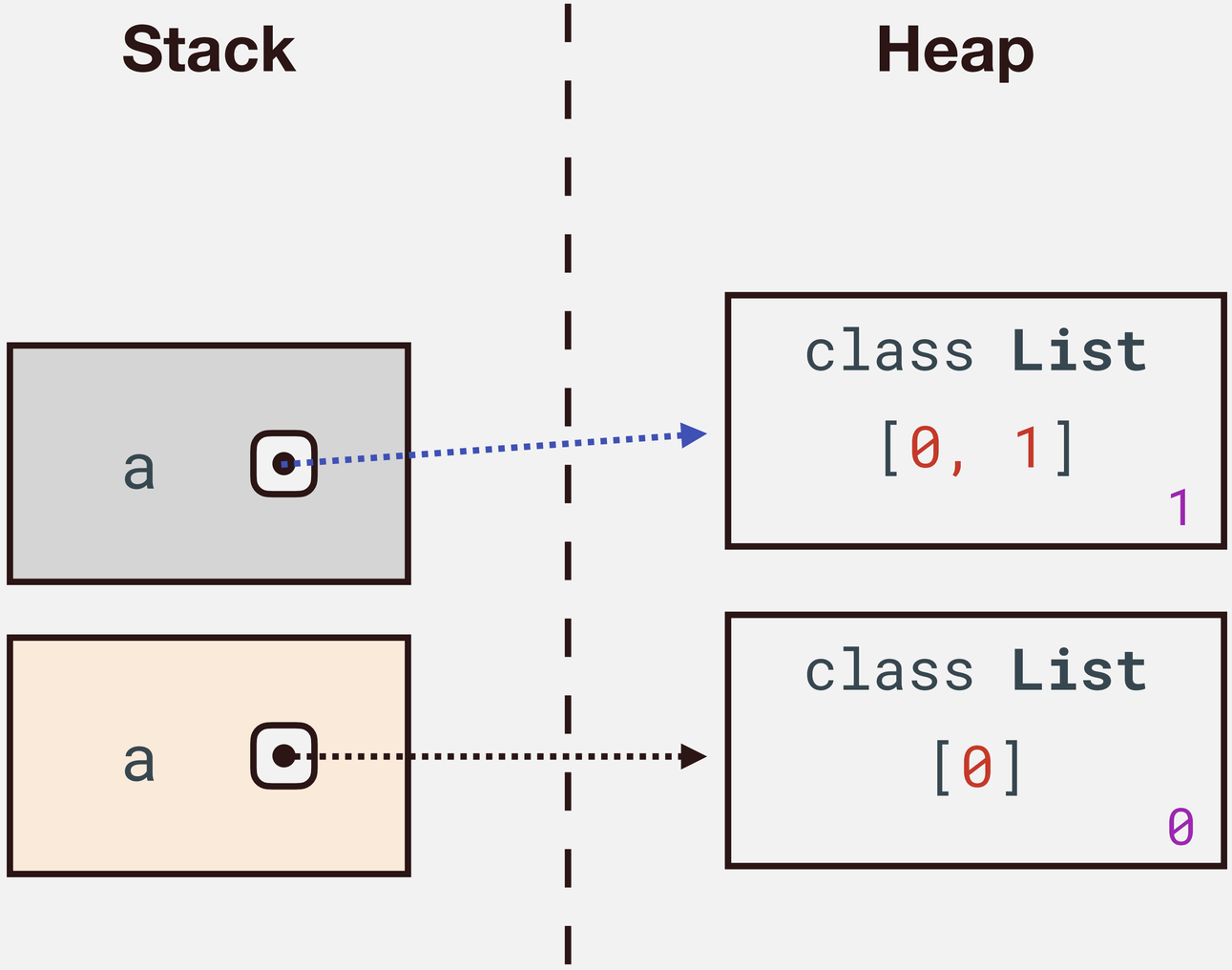
Pass by Object Reference

```
def reassign(a):
    a = [0, 1]

def append1(a):
    a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



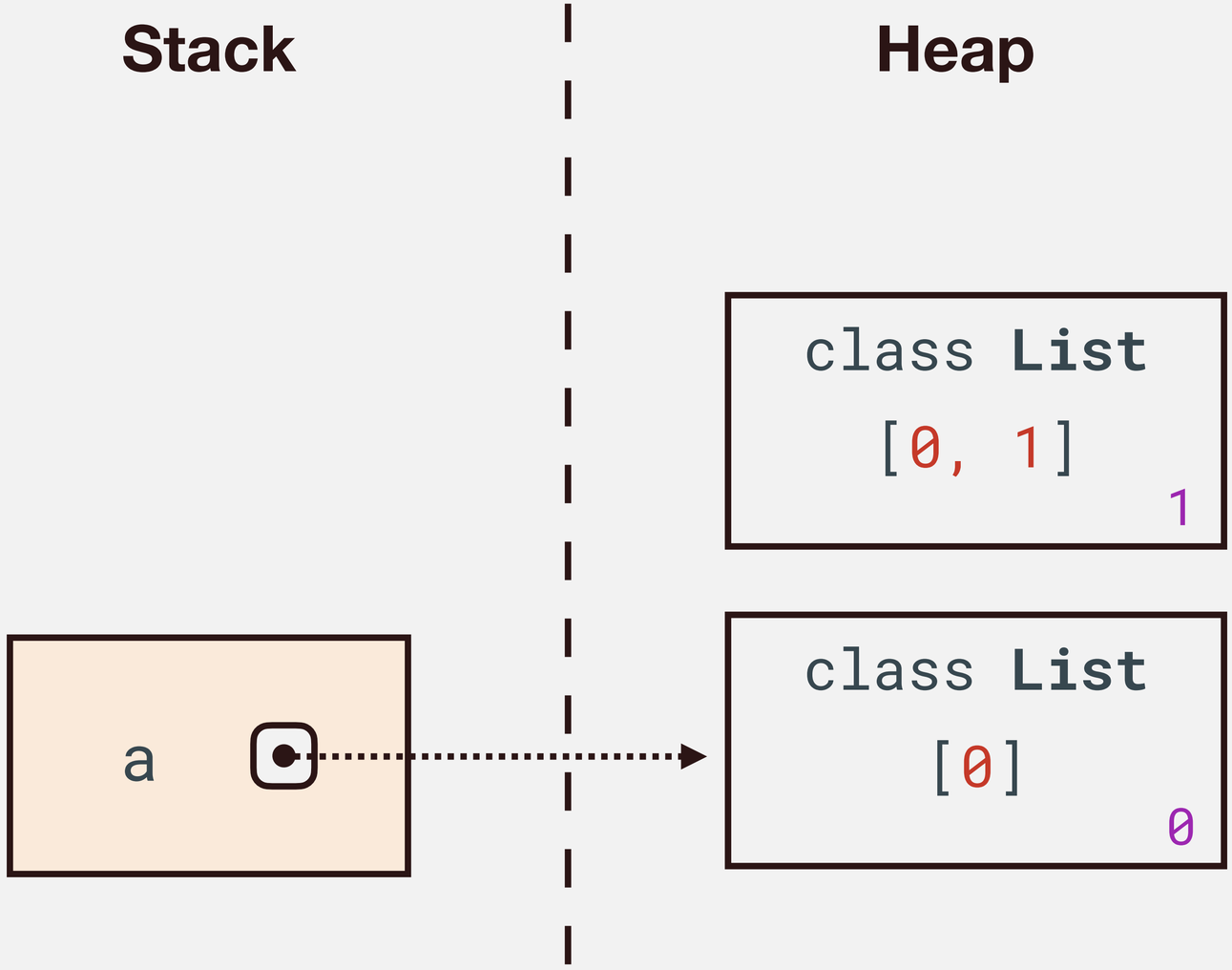
Pass by Object Reference

```
def reassign(a):
    a = [0, 1]

def append1(a):
    a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



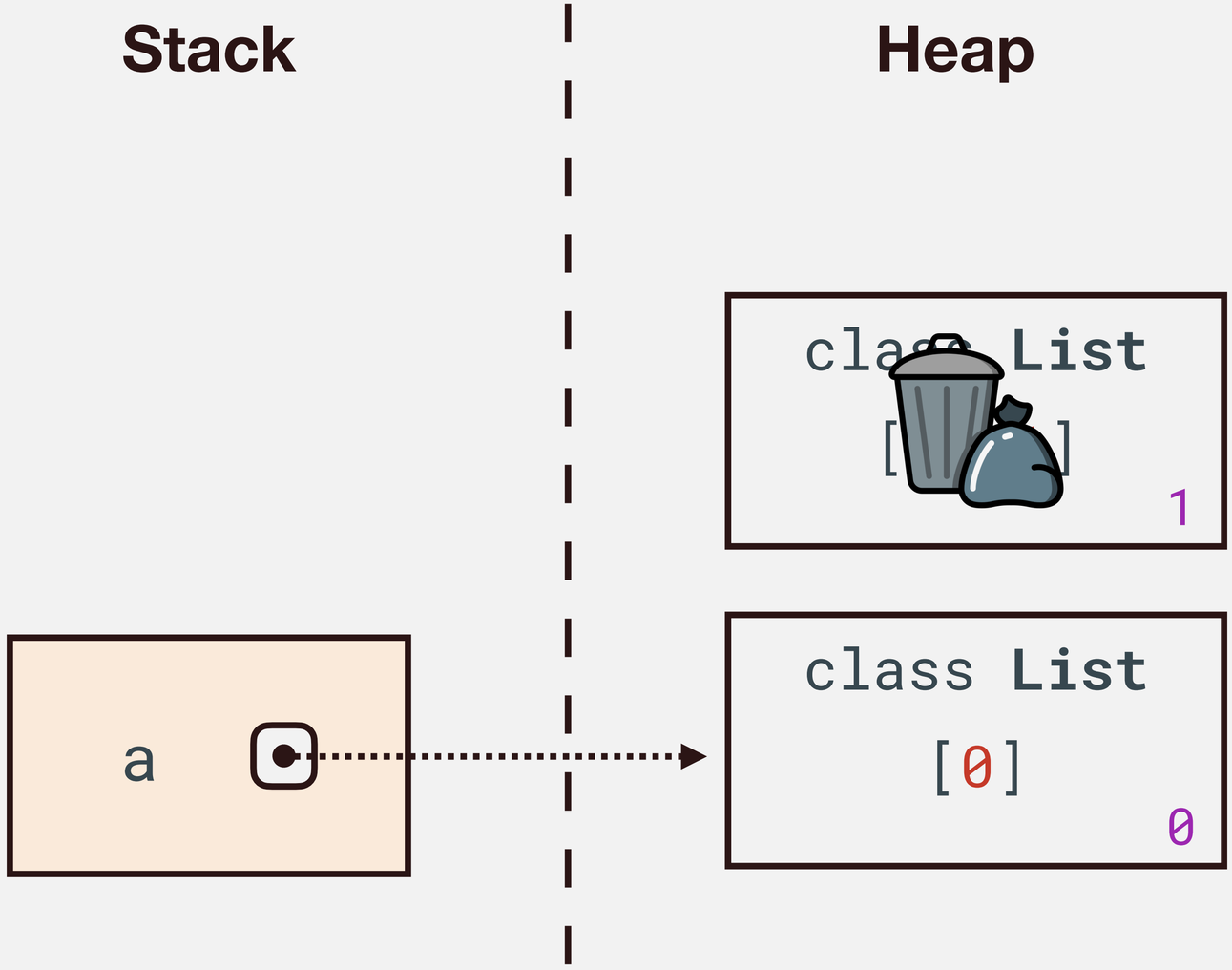
Pass by Object Reference

```
def reassign(a):
  a = [0, 1]

def append1(a):
  a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



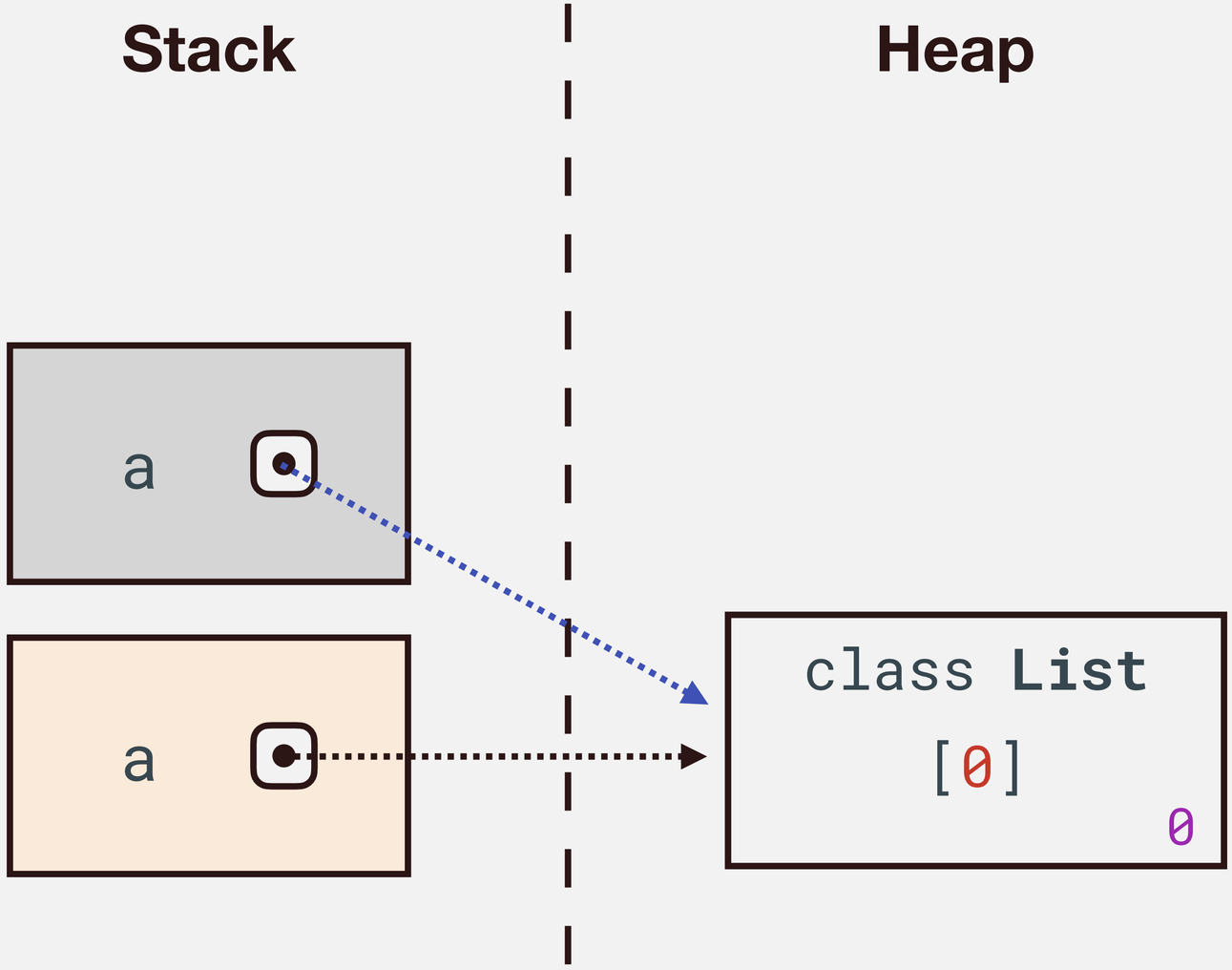
Pass by Object Reference

```
def reassign(a):
  a = [0, 1]

def append1(a):
  a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



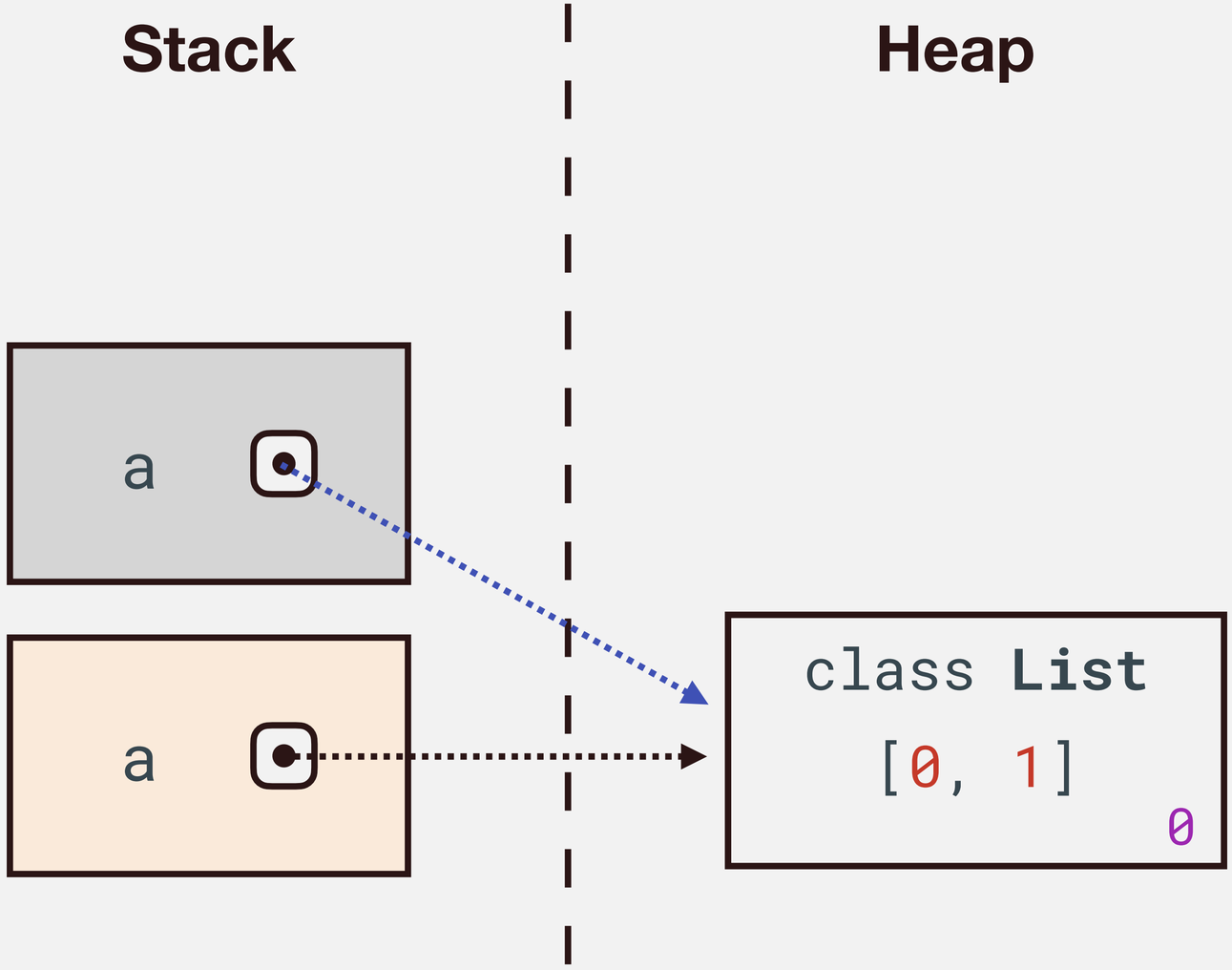
Pass by Object Reference

```
def reassign(a):
    a = [0, 1]

def append1(a):
    a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



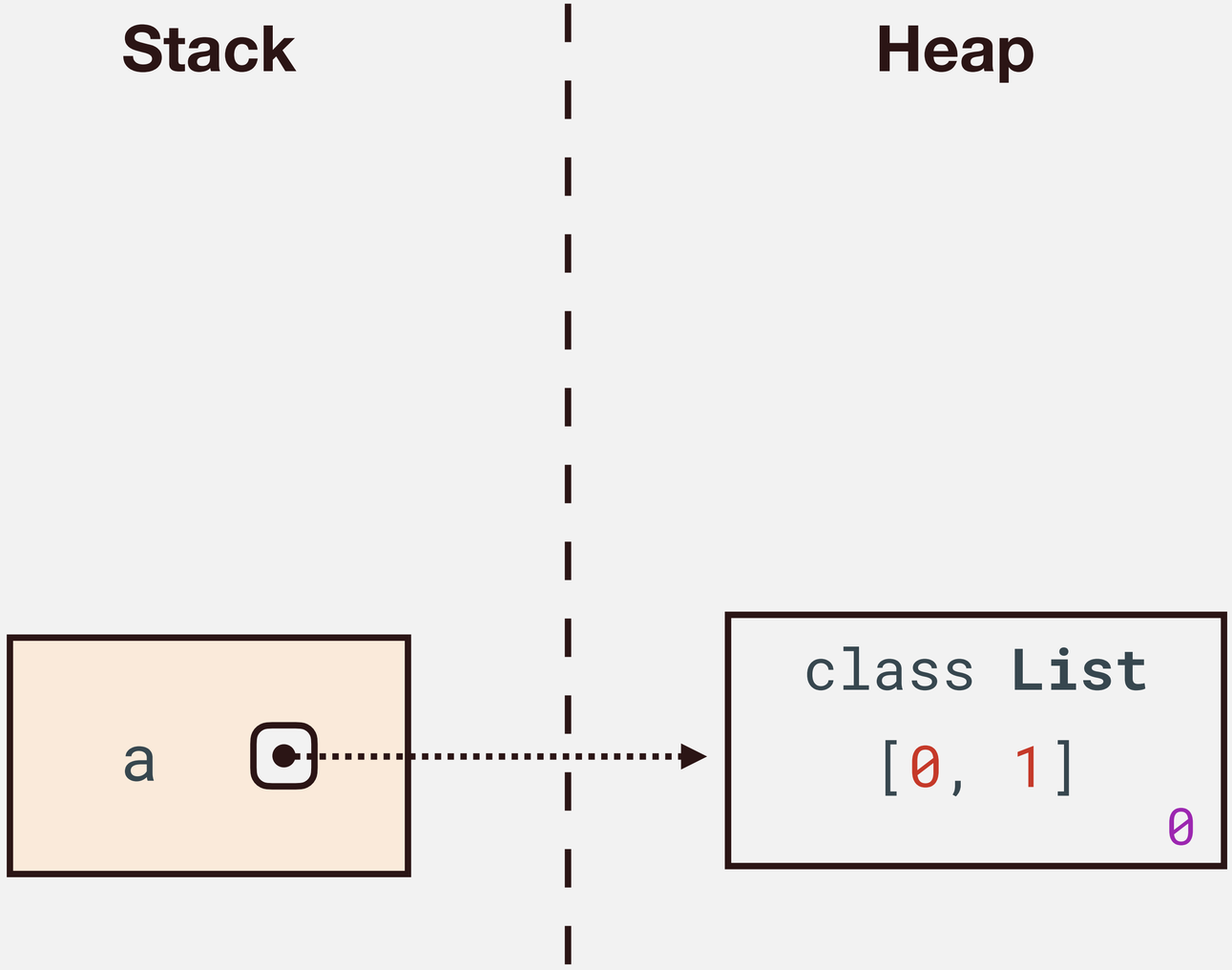
Pass by Object Reference

```
def reassign(a):
  a = [0, 1]

def append1(a):
  a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



Pass by Object Reference

```
def reassign(a):
  a = [0, 1]

def append1(a):
  a.append(1)

a = [0]
reassign(a)
append1(a)
print(a)
```



Default Arguments

```
def plot(x, y, color, marker, markersize,
         linestyle, linewidth):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .
```

Default Arguments

```
def plot(x, y, color, marker, markersize,
         linestyle, linewidth):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis, ‘black’, ‘o’, 10, ‘dashed’, 2)
```

Default Arguments

```
def plot(x, y, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis, 'black', 'o', 10, 'dashed', 2)
```

Default Arguments

```
def plot(x, y, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis)
```

Default Arguments

```
def plot(color='black', x, y, marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis)
```

Keyword Arguments

```
def plot(x, y, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis, color='blue', linewidth=3)
```

Keyword Arguments

```
def plot(x, y, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(color='blue', x_axis, y_axis, linewidth=3)
```

Arbitrary # of Positional Arguments

```
def plot(x, y, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis, color='blue', linewidth=3)
```

Arbitrary # of Positional Arguments

```
def plot(x, y, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(x, y):
    plot_point(p)
  . . .

plot(x_axis, y_axis, z_axis, color='blue', linewidth=3)
```

Arbitrary # of Positional Arguments

```
def plot(*args, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(*list(args)):
    plot_point(p)
  . . .

plot(x_axis, y_axis, z_axis, color='blue', linewidth=3)
```

unpack arguments

Arbitrary # of Keyword Arguments

```
def plot(*args, color='black', marker='o', markersize=10,
         linestyle='dashed', linewidth=2):
  set_color(color)
  set_marker(marker)
  . . .
  for p in zip(*list(args)):
    plot_point(p)
  . . .

plot(x_axis, y_axis, z_axis, color='blue', linewidth=3)
```

Arbitrary # of Keyword Arguments

```
def plot(*args, **kwargs):
  for k, v in kwargs.items():
    if k == ‘color’:
      set_color(v)
    elif k == ‘marker’:
      set_marker(v)
    . . .
  for p in zip(*list(args)):
    plot_point(p)

plot(x_axis, y_axis, z_axis, color=‘blue’, marker=‘o’)
```

Object-Oriented Programming

Constructors

C++

```
class Date {
  Date(int d, int m, int y)
    : day(d), month(m), year(y) {}

  int day, month, year;
};
```

Constructors

C++

```
class Date {
  Date(int d, int m, int y)
    : day(d), month(m), year(y) {}

  int day, month, year;
};
```

Python

```
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y
```

Constructors

C++

```
class Date {
  Date(int d, int m, int y)
    : day(d), month(m), year(y) {}

  int day, month, year;
};
```

Python

```
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y
```

like *this in C++
indicate member function

Constructors

C++

```cpp
class Date {
  Date(int d, int m, int y)
    : day(d), month(m), year(y) {}

  int day, month, year;
};

int main() {
  Date date = new Date(1, 1, 2022);
  date.day = 2;
}
```

Python

```python
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y

date = Date(1, 1, 2022)
date.day = 2
```

Member Functions

C++

```cpp
class Date {
  Date(int d, int m, int y)
    : day(d), month(m), year(y) {}
  int GetMonth() { return month; }
  void SetMonth(int m) {
    if (month > 0 && month < 13)
      month = m;
  }

  int day, month, year;
};
```

Python

```python
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y

  def get_month(self):
    return self.month

  def set_month(self, m):
    if self.month in range(1, 13):
      self.month = m
```

Encapsulation

C++

```cpp
class Date {
 public:
  Date(int d, int m, int y)
     : day(d), month(m), year(y) {}
  int GetMonth() { return month; }
  void SetMonth(int m) {
    if (month > 0 && month < 13)
      month = m;
  }
 private:
  int day, month, year;
};
```

Python

```python
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y

  def get_month(self):
    return self.month

  def set_month(self, m):
    if self.month in range(1, 13):
      self.month = m
```

Encapsulation

C++

```cpp
class Date {
 public:
  Date(int d, int m, int y)
     : day(d), month(m), year(y) {}
  int GetMonth() { return month; }
  void SetMonth(int m) {
    if (month > 0 && month < 13)
      month = m;
  }
 private:
  int day, month, year;
};
```

Python

```python
class Date:
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y

  def get_month(self):
    return self.__month

  def set_month(self, m):
    if self.__month in range(1, 13):
      self.__month = m
```

Encapsulation

C++

```
class Date {
 public:
  Date(int d, int m, int y)
     : day(d), month(m), year(y) {}
  int GetMonth() { return month; }
  void SetMonth(int m) {
    if (month > 0 && month < 13)
      month = m;
  }
 private:
  int day, month, year;
};
```

Python

```
class Date:
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y
```

It's OK, but NOT the Python way!

```
  def get_month(self):
    return self.__month

  def set_month(self, m):
    if self.__month in range(1, 13):
      self.__month = m
```

Encapsulation

```
class Date:
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y

  def get_month(self):
    return self.__month

  def set_month(self, m):
    if self.__month in range(1, 13):
      self.__month = m
```

```
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y
  @property
  def month(self):
    return self._month
  @month.setter
  def month(self, m):
    if self._month in range(1, 13):
      self._month = m
```

Encapsulation

```
class Date:
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y

  def get_month(self):
    return self.__month

  def set_month(self, m):
    if self.__month in range(1, 13):
      self.__month = m
```

```
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y
  @property
  def month(self):
    return self._month
  @month.setter
  def month(self, m):
    if self._month in range(1, 13):
      self._month = m
```

Decorator

Encapsulation

```
class Date:
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y

  def get_month(self):
    return self.__month

  def set_month(self, m):
    if self.__month in range(1, 13):
      self.__month = m
```

```
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y
  @property
  def month(self):
    return self._month
  @month.setter
  def month(self, m):
    if self._month in range(1, 13):
      self._month = m
```

Decorator

NOT required

Python Decorator

➜ A decorator function wraps an existing function and modifies its behavior

```
def my_decorator(f):
  def wrapper():
    f()
    print('is awesome!')
  return wrapper

def cpp():
  print('C++')
```

Python Decorator

➜ A decorator function wraps an existing function and modifies its behavior

```
def my_decorator(f):
  def wrapper():
    f()
    print('is awesome!')
  return wrapper

def cpp():
  print('C++')

cpp = my_decorator(cpp)
cpp() # C++ is awesome!
```

Python Decorator

➜ A decorator function wraps an existing function and modifies its behavior

```
def my_decorator(f):
  def wrapper():
    f()
    print('is awesome!')
  return wrapper

def cpp():
  print('C++')

cpp = my_decorator(cpp)
cpp() # C++ is awesome!
```

```
def my_decorator(f):
  def wrapper():
    f()
    print('is awesome!')
  return wrapper

@my_decorator
def cpp():
  print('C++')

cpp() # C++ is awesome!
```

Encapsulation

```
class Date:
  def __init__(self, d, m, y):
    self.day = d
    self.month = m
    self.year = y

  @property
  def month(self):
    return self._month

  @month.setter
  def month(self, m):
    if self._month in range(1, 13):
      self._month = m
```

```
date = Date(1, 1, 2022)
print(f'month = {date.month}')
date.month = 11
date.month = 100
```

Inheritance

```
class Base {
 public:
  Base(int id) : id_(id) {};
  virtual void output() const {
    std::cout << “id = ” << id_ << “\n”; }
 protected:
  int id_;
};
class Derived : public Base {
 public:
  Derived(int id, long c)
    : Base(id), count_(c) {};
  void output() override const {
    Base::output();
    std::cout << “c = ” << count_ << “\n”; }
 private:
  long count_;
};
```

Inheritance

```cpp
class Base {
 public:
  Base(int id) : id_(id) {};
  virtual void output() const {
    std::cout << “id = ” << id_ << “\n”; }
 protected:
  int id_;
};
class Derived : public Base {
 public:
  Derived(int id, long c)
    : Base(id), count_(c) {};
  void output() override const {
    Base::output();
    std::cout << “c = ” << count_ << “\n”; }
 private:
  long count_;
};
```

```python
class Base:
  def __init__(self, id):
    self.__id = id
  def output(self):
    print(f‘id = {self.__id}’)

class Derived(Base):
  def __init__(self, id, count):
    super().__init__(id)
    self.__count = count
  def output(self):
    super().output()
    print(f‘c = {self.__count}’)
```

Overloading

C++

```
class Vec {
 public:
  Vec(int x, int y) : x_(x), y_(y) {}
  friend Vec operator+(const Vec &a,
                       int val);
 private:
  int x_, y_;
};

Vec operator+(const Vec &a,
              const Vec &b) {
  return Vec(a.x_ + b.x_, a.y_ + b.y_);
}
```

Overloading

C++

```
class Vec {
 public:
  Vec(int x, int y) : x_(x), y_(y) {}
  friend Vec operator+(const Vec &a,
                       int val);
 private:
  int x_, y_;
};

Vec operator+(const Vec &a,
              const Vec &b) {
  return Vec(a.x_ + b.x_, a.y_ + b.y_);
}
```

Python

```
class Vec:
  def __init__(self, x, y):
    self.__x = x
    self.__y = y

  def __add__(self, other):
    return Vec(self.__x + other.x,
               self.__y + other.y)

v1 = Vec(1, 2)
v2 = Vec(3, 4)
v3 = v1 + v2
```

Python Magic/Dunder Methods

➜ Invocation happens internally from the class on certain actions

Built-in	Dunder	Built-in	Dunder
`+`	`__add__(self, other)`	`<`	`__lt__(self, other)`
`-`	`__sub__(self, other)`	`<=`	`__le__(self, other)`
`*`	`__mul__(self, other)`	`>`	`__gt__(self, other)`
`/`	`__floordiv__(self, other)`	`>=`	`__ge__(self, other)`
`//`	`__truediv__(self, other)`	`abs()`	`__abs__(self)`
`%`	`__mod__(self, other)`	`int()`	`__int__(self)`
`**`	`__pow__(self, other)`	`str()`	`__str__(self)`
`==`	`__eq__(self, other)`	`len()`	`__len__(self)`
`!=`	`__ne__(self, other)`	• • •	

Overloading

```
class Vec:
  def __init__(self, x, y):
    self.__x = x
    self.__y = y

  def __add__(self, other):
    return Vec(self.__x + other.x,
               self.__y + other.y)

  def __str__(self):
    return f'({self.__x}, {self__y})'
```

Python Magic/Dunder Methods

➜ Invocation happens internally from the class on certain actions

Action	Dunder
Create an object	`__new__(cls, …)`
Initialize an object, called by __new__	`__init__(self, …)`
called when an accessing attribute does NOT exist	`__getattr__(self, name)`

• • •

Static Members

C++

```cpp
class Date {
  static int date_count = 0;
 public:
  Date(int d, int m, int y)
     : day_(d), month_(m), year_(y) {
    date_count++;
  }
  static int GetDateCount() {
    return date_count;
  }
 private:
  int day_, month_, year_;
};
```

Python

```python
class Date:
  date_count = 0
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y
    Date.date_count += 1

  @classmethod
  def get_date_count(cls):
    return cls.date_count
```

Static Members

```
class Date:
  date_count = 0
  def __init__(self, d, m, y):
    self.__day = d
    self.__month = m
    self.__year = y
    Date.date_count += 1

  @classmethod
  def get_date_count(cls):
    return cls.date_count

  @staticmethod
  def inc(num):
    return num + 1
```

Exceptions

Runtime Errors

```
n = int(input('Please enter a number: '))
print(f'n = {n}')

>>> Please Enter a number:
```

Runtime Errors

```
n = int(input('Please enter a number: '))
print(f'n = {n}')

>>> Please Enter a number: No
```

Runtime Errors

```
n = int(input('Please enter a number: '))
print(f'n = {n}')

>>> Please Enter a number: No
>>> Traceback
    . . .
    ValueError: invalid literal for int() with base 10: 'No'
```

try-except-else

```
try:
  n = int(input('Please enter a number: '))
except ValueError:
  print('Invalid n')
else:
  print(f'n = {n}')
```

try-except-else

```
def get_user_int(prompt):
  while True:
    try:
      return int(input(prompt))
    except ValueError:
      pass

n = get_user_int('Please enter a number: ')
print(f'n = {n}')
```

Common Built-in Errors

Error	Description
`SyntaxError`	Parser detects syntax error
`IndexError`	Index out of bound
`NameError`	Variable name not found
`KeyError`	Key is not found in a dictionary
`ValueError`	A function gets an improper value as argument
`ZeroDivisionError`	Divided by zero
`OSError`	A system function causes system-related error
• • •	https://docs.python.org/3/library/exceptions.html#bltin-exceptions

Finally

```
. . .
try:
  n = x / y
except ZeroDivisionError:
  print('Divided by 0!')
else:
  print(f'n = {n}')
finally:
  close_files()
  close_connections()
  . . .
```

Basic File I/O

Writing to a File

```
While True:
  name = input('Your name, please: ')
  if not name:
    break
  file = open('names.txt', 'w')
  file.write(f'{name}\n')
  file.close()
```

Writing to a File

```
While True:
  name = input('Your name, please: ')
  if not name:
    break
  file = open('names.txt', 'w')
  file.write(f'{name}\n')
  file.close()
```

Writing to a File

```
While True:
  name = input('Your name, please: ')
  if not name:
    break
  file = open('names.txt', 'a')
  file.write(f'{name}\n')
  file.close()
```

append to file

Writing to a File

```
While True:
  name = input('Your name, please: ')
  if not name:
    break
  with open('names.txt', 'a') as file:
    file.write(f'{name}\n')
```

Reading a CSV File

```
faculty = {}
with open('research.csv', 'r') as file:
  for line in file:
    row = line.rstrip().split(',')
    faculty[row[0]] = row[1]
```

research.csv

Name, **Field**

Andrew Yao, Everything

Wei Xu, Distributed Systems

Longbo Huang, Network and AI

Jian Li, Theory

Ran Duan, Theory

Minyu Gao, Architecture

Yi Wu, Reinforcement Learning

Huanchen Zhang, Database Systems

Reading a CSV File

```
import csv
faculty = {}
with open('research.csv', 'r') as file:
  reader = csv.reader(file):
  for row in reader:
    faculty[row[0]] = row[1]
```

research.csv

Name, **Field**

Andrew Yao, Everything

Wei Xu, Distributed Systems

Longbo Huang, Network and AI

Jian Li, Theory

Ran Duan, Theory

Minyu Gao, Architecture

Yi Wu, Reinforcement Learning

Huanchen Zhang, Database Systems

Reading a CSV File

```
import csv
faculty = {}
with open('research.csv', 'r') as file:
  reader = csv.DictReader(file):
  for row in reader:
    faculty[row['Name']] = row['Field']
```

research.csv

Name, **Field**

Andrew Yao, Everything

Wei Xu, Distributed Systems

Longbo Huang, Network and AI

Jian Li, Theory

Ran Duan, Theory

Minyu Gao, Architecture

Yi Wu, Reinforcement Learning

Huanchen Zhang, Database Systems

Libraries/Modules/Packages

Powerful Libraries make Python popular

- ➜ Python Standard Library: e.g., `math, os, sys`
- ➜ Third-party Library: e.g., `numpy, sklearn`

Powerful Libraries make Python popular

- Python Standard Library: e.g., `math`, `os`, `sys`
- Third-party Library: e.g., `numpy`, `sklearn`

```
import math
math.sqrt(16)
math.cos(math.pi / 3)
```

Powerful Libraries make Python popular

- Python Standard Library: e.g., `math`, `os`, `sys`
- Third-party Library: e.g., `numpy`, `sklearn`

```
import math
math.sqrt(16)
math.cos(math.pi / 3)


from math import sqrt, cos, pi
sqrt(16)
cos(pi / 3)
```

Powerful Libraries make Python popular

- Python Standard Library: e.g., `math`, `os`, `sys`
- Third-party Library: e.g., `numpy`, `sklearn`

```
import numpy as np
from sklearn import linear_model as lm

X = np.arange(10).reshape((-1, 1))
y = [2, 5, 6, 9, 10, 12, 14, 15, 19, 20]

reg = lm.LinearRegression()
reg.fit(X, y)
y_pred = reg.predict(X)
```