Python - Data abstraction & Type abstraction

June 14, 2018

1 Data abstraction & Type abstraction

1.1 Agenda

- Essential Concepts
  - Type
  - Type System
  - Polymorphism
- Python
  - Object
  - Type System
  - Class

```
In [5]: print(type(1))
        print(type(True))

        def foo(): pass

        print(type(foo))

        class A: pass
        class B(object): pass

        print(type(A))
        print(type(B))
        isinstance(B, type)
```

```
<type 'int'>
<type 'bool'>
<type 'function'>
<type 'classobj'>
<type 'type'>
```

```
Out[5]: True
```

1.2 Essential Concepts

1.3 Type

1.3.1 Why type?

- Types provide implicit context for many operations, so that the programmer does not have to specify that context explicitly.
- Types limit the set of operations that may be performed in a semantically valid program.
- If types are specified explicitly in the source program, they can often make the program easier to read and understand.
- If types are known at compile time, they can be used to drive important performance optimizations.

1.4 Type System

Informally, a type system consists of 1. a mechanism to define types and associate them with certain language constructs, and 2. a set of rules for type equivalence, type compatibility, and type inference.

Type Checking and Type Safety

- Type checking is the process of ensuring that a program obeys the language's type compatibility rules. A violation of the rules is known as a type clash.
- A language is said to be strongly typed if it prohibits, in a way that the language implementation can enforce, the application of any operation to any object that is not intended to support that operation.
- A language is said to be statically typed if it is strongly typed and type checking can be performed at compile time.

Type compatibility is the one of most concern to programmers. In many languages, however, compatibility is a looser relationship than equivalence: objects and contexts are often compatible even when their types are different.

Type equivalence There are two principal ways of defining type equivalence. - Structural equivalence is based on the content of type definitions. - Name equivalence is based on the lexical occurrence of type definitions.

Type compatibility Most languages do not require equivalence of types in every context. Instead, they merely say that a value's type must be compatible with that of the context in which it appears.

Universal Reference Types

```
// C
void *ref;
ref = &object;
```

```
# Python
hello = world

// javascript
var hello = world
```

One way to ensure the safety of universal to specific assignments is to make the objects self-descriptive——that is, to include in the representation of each object a tag that indicates its type.

coercion Type coercion is performed automatically and implicitly whenever a language allows a value of one type to be used in a context that expects another.

What if the coercion can not be performed?

1.4.1 Polymorphism

Parametric Polymorphism & Subtype Polymorphism

```
// C++: explicit parametric polymorphism
template <typename T>
T const& Max (T const& a, T const& b) {
    return a < b ? b:a;
}

-- haskell: implicit parametric polymorphism
let max a b = if a < b then b else a

# Python: duck typing
def max(a, b):
    return b if a < b else a

// C++: subtype polymorphism
BaseClass const& Max (BaseClass const& a, BaseClass const& b) {
    return a < b ? b:a;
}
```

According to different language implementations, - **What implicit context does the type system provide?** - **What operations does the type system limit?**

1.5 Python

1.6 Recap

1.6.1 Everything is an object in Python.

1.6.2 PyObject & PyVarObject

```
// https://github.com/python/cpython/blob/2.7/Include/object.h#L106
typedef struct _object {
    PyObject_HEAD
} PyObject;
```

```
typedef struct {
    PyObject_VAR_HEAD
} PyVarObject;
```

1.6.3 PyObject_HEAD

```
// https://github.com/python/cpython/blob/2.7/Include/object.h#L78
/* PyObject_HEAD defines the initial segment of every PyObject. */
#define PyObject_HEAD                       \
    _PyObject_HEAD_EXTRA                    \
    Py_ssize_t ob_refcnt;                   \
    struct _typeobject *ob_type;

// https://github.com/python/cpython/blob/2.7/Include/object.h#L64
#ifdef Py_TRACE_REFS
/* ... */
#else
#define _PyObject_HEAD_EXTRA
#define _PyObject_EXTRA_INIT
#endif
```

1.6.4 PyObject_VAR_HEAD

```
// https://github.com/python/cpython/blob/2.7/Include/object.h#L96
/* PyObject_VAR_HEAD defines the initial segment of all variable-size
 * container objects. */
#define PyObject_VAR_HEAD                   \
    PyObject_HEAD                           \
    Py_ssize_t ob_size; /* Number of items in variable part */
```

1.7 Object

- Objects are Python's abstraction for data.
- Every object has an identity, a type and a value.

- isinstance
- issubclass

https://github.com/python/cpython/blob/2.7/Objects/abstract.c#L2939

1.8 Type

1.8.1 What on earth is the type?

```
In [2]: x = 0b01000001
        chr(x), int(x)

Out[2]: ('A', 65)
```

1.8.2 Comparison of the type system

Language	Type Safety	Type Expression	Type compatibility and equivalence	Type checking
Python	Strong	implicit (with optional explicit typing as of 3.5)	structural	dynamic
C	weak	explicit	nominal	static
Java	strong	explicit	nominal	static

https://en.wikipedia.org/wiki/Comparison_of_programming_languages_by_type_system

1.8.3 New-style and classic classes

```
In [9]: class A: pass
        class B: pass
        a = A()
        b = B()
        type(a) is type(b)

Out[9]: True
```

```
In [4]: class A(object): pass
        class B(object): pass
        a = A()
        b = B()
        type(a) is type(b)

Out[4]: False
```

1.8.4 New-style and classic classes

- Up to Python 2.1 the concept of class was unrelated to the concept of type, and old-style classes were the only flavor available.
- New-style classes were introduced in Python 2.2 to unify the concepts of class and type.
- The major motivation for introducing new-style classes is to provide a unified object model with a full meta-model. It also has a number of practical benefits, like the ability to subclass most built-in types, or the introduction of "descriptors", which enable computed properties.
- For compatibility reasons, classes are still old-style by default. New-style classes are created by specifying another new-style class (i.e. a type) as a parent class, or the "top-level type" object if no other parent is needed.
- Old-style classes are removed in Python 3, leaving only new-style classes.
- https://docs.python.org/2/reference/datamodel.html#new-style-and-classic-classes
- http://python-history.blogspot.sg/2010/06/inside-story-on-new-style-classes.html

1.8.5 The first commit to introduce new-style classes

https://github.com/python/cpython/commit/6d6c1a35e08b95a83dbe47dbd9e6474daff00354

1.8.6 PyTypeObject in detail

```c
// https://github.com/python/cpython/blob/2.7/Include/object.h#L324
// https://docs.python.org/2/c-api/typeobj.html
typedef struct _typeobject {
    PyObject_VAR_HEAD
    const char *tp_name; /* For printing, in format "<module>.<name>" */
    Py_ssize_t tp_basicsize, tp_itemsize; /* For allocation */

    /* Methods to implement standard operations */

    destructor tp_dealloc;
    printfunc tp_print;
    getattrfunc tp_getattr;
    setattrfunc tp_setattr;
    cmpfunc tp_compare;
    reprfunc tp_repr;

    /* Method suites for standard classes */

    PyNumberMethods *tp_as_number;
    PySequenceMethods *tp_as_sequence;
    PyMappingMethods *tp_as_mapping;

    /* More standard operations (here for binary compatibility) */

    hashfunc tp_hash;
    ternaryfunc tp_call;
    reprfunc tp_str;
    getattrofunc tp_getattro;
    setattrofunc tp_setattro;

    /* Functions to access object as input/output buffer */
    PyBufferProcs *tp_as_buffer;

    /* Flags to define presence of optional/expanded features */
    long tp_flags;

    const char *tp_doc; /* Documentation string */

    /* Assigned meaning in release 2.0 */
    /* call function for all accessible objects */
    traverseproc tp_traverse;
    /* delete references to contained objects */
    inquiry tp_clear;

    /* Assigned meaning in release 2.1 */
    /* rich comparisons */
    richcmpfunc tp_richcompare;

    /* weak reference enabler */
    Py_ssize_t tp_weaklistoffset;

    /* Added in release 2.2 */
    /* Iterators */
    getiterfunc tp_iter;
    iternextfunc tp_iternext;

    /* Attribute descriptor and subclassing stuff */
    struct PyMethodDef *tp_methods;
    struct PyMemberDef *tp_members;
    struct PyGetSetDef *tp_getset;
    struct _typeobject *tp_base;
    PyObject *tp_dict;
    descrgetfunc tp_descr_get;
    descrsetfunc tp_descr_set;
    Py_ssize_t tp_dictoffset;
    initproc tp_init;
    allocfunc tp_alloc;
    newfunc tp_new;
    freefunc tp_free; /* Low-level free-memory routine */
    inquiry tp_is_gc; /* For PyObject_IS_GC */
    PyObject *tp_bases;
    PyObject *tp_mro; /* method resolution order */
    PyObject *tp_cache;
    PyObject *tp_subclasses;
    PyObject *tp_weaklist;
    destructor tp_del;

    /* Type attribute cache version tag. Added in version 2.6 */
    unsigned int tp_version_tag;

#ifdef COUNT_ALLOCS
    /* these must be last and never explicitly initialized */
    Py_ssize_t tp_allocs;
    Py_ssize_t tp_frees;
    Py_ssize_t tp_maxalloc;
    struct _typeobject *tp_prev;
    struct _typeobject *tp_next;
#endif
} PyTypeObject;
```

General functions and operators

- printfunc tp_print;
- reprfunc tp_repr;
- reprfunc tp_str;
- hashfunc tp_hash;
- . . .

callable ternaryfunc tp_call;

Protocols

- Number protocol
  - PyNumberMethods *tp_as_number;
- Mapping protocol (tp_as_mapping)
  - PyMappingMethods *tp_as_mapping;
- Sequence protocol (tp_as_sequence)
  - PySequenceMethods *tp_as_sequence;
- Buffer protocol (tp_as_buffer)
  - PyBufferProcs *tp_as_buffer;

The methods, members & descriptors of types

- Methods
  - struct PyMethodDef *tp_methods;
- Members
  - struct PyMemberDef *tp_members;
- Descriptors
  - struct PyGetSetDef *tp_getset;

all above will be in PyObject *tp_dict;

Instance dict offset Py_ssize_t tp_dictoffset;

metatype

```
// https://github.com/python/cpython/blob/2.7/Include/object.h#L441
PyAPI_DATA(PyTypeObject) PyType_Type; /* built-in 'type' */

// https://github.com/python/cpython/blob/2.7/Objects/typeobject.c#L2870
PyTypeObject PyType_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0)
```

```c
    "type",                                     /* tp_name */
    sizeof(PyHeapTypeObject),                   /* tp_basicsize */
    sizeof(PyMemberDef),                        /* tp_itemsize */
    (destructor)type_dealloc,                   /* tp_dealloc */
    0,                                          /* tp_print */
    0,                                          /* tp_getattr */
    0,                                          /* tp_setattr */
    0,                                  /* tp_compare */
    (reprfunc)type_repr,                        /* tp_repr */
    0,                                          /* tp_as_number */
    0,                                          /* tp_as_sequence */
    0,                                          /* tp_as_mapping */
    (hashfunc)_Py_HashPointer,                  /* tp_hash */
    (ternaryfunc)type_call,                     /* tp_call */
    0,                                          /* tp_str */
    (getattrofunc)type_getattro,                /* tp_getattro */
    (setattrofunc)type_setattro,                /* tp_setattro */
    0,                                          /* tp_as_buffer */
    Py_TPFLAGS_DEFAULT | Py_TPFLAGS_HAVE_GC |
        Py_TPFLAGS_BASETYPE | Py_TPFLAGS_TYPE_SUBCLASS,         /* tp_flags */
    type_doc,                                   /* tp_doc */
    (traverseproc)type_traverse,                /* tp_traverse */
    (inquiry)type_clear,                        /* tp_clear */
    type_richcompare,                                           /* tp_richcompare */
    offsetof(PyTypeObject, tp_weaklist),        /* tp_weaklistoffset */
    0,                                          /* tp_iter */
    0,                                          /* tp_iternext */
    type_methods,                               /* tp_methods */
    type_members,                               /* tp_members */
    type_getsets,                               /* tp_getset */
    0,                                          /* tp_base */
    0,                                          /* tp_dict */
    0,                                          /* tp_descr_get */
    0,                                          /* tp_descr_set */
    offsetof(PyTypeObject, tp_dict),            /* tp_dictoffset */
    type_init,                                  /* tp_init */
    0,                                          /* tp_alloc */
    type_new,                                   /* tp_new */
    PyObject_GC_Del,                            /* tp_free */
    (inquiry)type_is_gc,                        /* tp_is_gc */
};
```

User defined class

```c
/* The *real* layout of a type object when allocated on the heap */
typedef struct _heaptypeobject {
    /* Note: there's a dependency on the order of these members
       in slotptr() in typeobject.c . */
    PyTypeObject ht_type;
    PyNumberMethods as_number;
    PyMappingMethods as_mapping;
    PySequenceMethods as_sequence; /* as_sequence comes after as_mapping,
                                      so that the mapping wins when both
                                      the mapping and the sequence define
                                      a given operator (e.g. __getitem__).
                                      see add_operators() in typeobject.c . */
    PyBufferProcs as_buffer;
    PyObject *ht_name, *ht_slots;
    /* here are optional user slots, followed by the members. */
} PyHeapTypeObject;
```

1.8.7 Understand descriptor

- Descriptor HowTo Guide: https://docs.python.org/2/howto/descriptor.html
- _PyObject_GenericGetAttrWithDict: https://github.com/python/cpython/blob/2.7/Objects/object.c#L13
- _PyType_Lookup: https://github.com/python/cpython/blob/2.7/Objects/typeobject.c#L2523

mro

- https://stackoverflow.com/questions/15404256/changing-type-of-python-objects
- Builtin-methods: https://github.com/python/cpython/blob/2.7/Python/bltinmodule.c#L2615
- _PyBuiltin_Init: https://github.com/python/cpython/blob/2.7/Python/bltinmodule.c#L2678
- PyAPI_FUNC & PyAPI_DATA: https://github.com/python/cpython/blob/2.7/Include/pyport.h#L746
- PyType_Type declaration: https://github.com/python/cpython/blob/2.7/Include/object.h#L441
- PyType_Type assignment: https://github.com/python/cpython/blob/2.7/Objects/typeobject.c#L2870
- PyBaseObject_Type: https://github.com/python/cpython/blob/2.7/Objects/typeobject.c#L3672
- classmethod & staticmethod: https://github.com/python/cpython/blob/2.7/Objects/funcobject.c
- Type annotation: https://www.youtube.com/watch?v=2wDvzy6Hgxg
- Float point number: https://en.wikipedia.org/wiki/IEEE_754
- Inheritance is not subtyping: https://dl.acm.org/citation.cfm?id=96721