Types

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 1

The role of types

- Interpreting data stored in binary in a consistent manner
  - View sequence of bits as integers, floats, characters, ...
  - Nature and range of allowed values
  - Operations that are permitted on these values

The role of types

- Interpreting data stored in binary in a consistent manner
  - View sequence of bits as integers, floats, characters, ...
  - Nature and range of allowed values
  - Operations that are permitted on these values
- Naming concepts and structuring our computation
  - Especially at a higher level
  - `Point` vs `(Float,Float)`
  - Banking application: accounts of different types, customers ...

The role of types

- Interpreting data stored in binary in a consistent manner
  - View sequence of bits as integers, floats, characters, ...
  - Nature and range of allowed values
  - Operations that are permitted on these values
- Naming concepts and structuring our computation
  - Especially at a higher level
  - `Point` vs `(Float,Float)`
  - Banking application: accounts of different types, customers ...
- Catching bugs early
  - Incorrect expression evaluation — like dimension mismatch in science
  - Incorrect assignment — expression value does not match variable type

Dynamic vs static typing

- Every variable we use has a type
- How is the type of a variable determined?

Dynamic vs static typing

- Every variable we use has a type
- How is the type of a variable determined?
- Python determines the type based on the current value
  - Dynamic typing — names derive type from current value
  - `x = 10` — `x` is of type `int`
  - `x = 7.5` — now `x` is of type `float`
  - An uninitialized name as no type

Dynamic vs static typing

- Every variable we use has a type
- How is the type of a variable determined?
- Python determines the type based on the current value
  - Dynamic typing — names derive type from current value
  - `x = 10` — `x` is of type `int`
  - `x = 7.5` — now `x` is of type `float`
  - An uninitialized name as no type
- Static typing — associate a type in advance with a name
  - Need to declare names and their types in advance value
  - `int x`, `float a`, ...
  - Cannot assign an incompatible value — `x = 7.5` is no longer legal

Dynamic vs static typing

- "Isn't it convenient that we don't have to declare variables in advance in Python?"
- Yes, but ...

Dynamic vs static typing

- "Isn't it convenient that we don't have to declare variables in advance in Python?"
- Yes, but ...
- Difficult to catch errors, such as typos

```
def factors(n):
  factorlist = []
  for i in range(1,n+1):
    if n%i == 0:
       factorlst = factorlist + [i]
  return(factorlist)
```

Dynamic vs static typing

- "Isn't it convenient that we don't have to declare variables in advance in Python?"
- Yes, but ...
- Difficult to catch errors, such as typos

```
def factors(n):
  factorlist = []
  for i in range(1,n+1):
    if n%i == 0:
       factorlst = factorlist + [i]  # Typo!
  return(factorlist)
```

- Empty user defined objects
  - Linked list is a sequence of objects of type `Node`
  - Convenient to represent empty linked list by `None`
  - Without declaring type of `l`, Python cannot associate a type after `l = None`

Types for organizing concepts

- Even simple type "synonyms" can help clarify code
  - 2D point is a pair `(float,float)`, 3D point is triple `(float,float,float)`
  - Create new type names `point2d` and `point3d`
  - These are synonyms for `(float,float)` and `(float,float,float)`
  - Makes intent more transparent when writing, reading and maintaining code

Types for organizing concepts

- Even simple type "synonyms" can help clarify code
  - 2D point is a pair `(float,float)`, 3D point is triple `(float,float,float)`
  - Create new type names `point2d` and `point3d`
  - These are synonyms for `(float,float)` and `(float,float,float)`
  - Makes intent more transparent when writing, reading and maintaining code
- More elaborate types — abstract datatypes and object-oriented programming
  - Consider a banking application
  - Data and operations related to accounts, customers, deposits, withdrawals, transfers
  - Denote accounts and customers as separate types
  - Deposits, withdrawals, transfers can be applied to accounts, not customers
  - Updating personal details applies to customers, not accounts

Static analysis

- Identify errors as early as possible — saves cost, effort

Static analysis

- Identify errors as early as possible — saves cost, effort
- In general, compilers cannot check that a program will work correctly
  - Halting problem — Alan Turing

Static analysis

- Identify errors as early as possible — saves cost, effort
- In general, compilers cannot check that a program will work correctly
  - Halting problem — Alan Turing
- With variable delarations, compilers can detect type errors at compile-time — static analysis
  - Dynamic typing would catch these errors only when the code runs
  - Executing code also slows down due to simultaneous monitoring for type correctness

Static analysis

- Identify errors as early as possible — saves cost, effort
- In general, compilers cannot check that a program will work correctly
  - Halting problem — Alan Turing
- With variable delarations, compilers can detect type errors at compile-time — static analysis
  - Dynamic typing would catch these errors only when the code runs
  - Executing code also slows down due to simultaneous monitoring for type correctness
- Compilers can also perform optimizations based on static analysis
  - Reorder statements to optimize reads and writes
  - Store previously computed expressions to re-use later

Summary

- Types have many uses
  - Making sense of arbitrary bit sequences in memory
  - Organizing concepts in our code in a meaningful way
  - Helping compilers catch bugs early, optimize compiled code
- Some languages also support automatic type inference
  - Deduce the types of variable statically, based on the context in which they are used
  - `x = 7` followed by `y = x + 15` implies `y` must be `int`
  - If the inferred type is consistent across the program, all is well