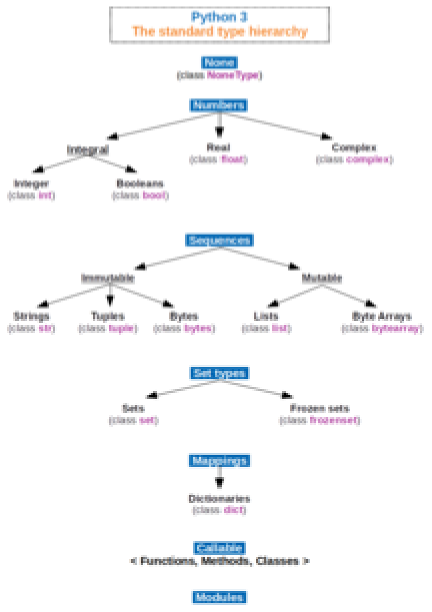
WIKIPEDIA

Data type

In computer science and computer programming, a **data type** or simply **type** is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data. Most programming languages support basic data types of integer numbers (of varying sizes), Floating-point numbers (which approximate real numbers), characters and Booleans. A data type constrains the values that an expression, such as a variable or a function, might take. This data type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored. A data type provides a set of values from which an expression (i.e. variable, function, etc.) may take its values.[1][2]



Python 3: the standard type hierarchy

Contents

Concept

Data types are used within type systems, which offer various ways of defining, implementing and using them. Different type systems ensure varying degrees of type safety.

Almost all programming languages explicitly include the notion of data type, though different languages may use different terminology.

Common data types include:

- Integer.
- Floating-point number.
- Character.
- String.
- Boolean.

For example, in the Java programming language, the type *int* represents the set of 32-bit integers ranging in value from −2,147,483,648 to 2,147,483,647, as well as the operations that can be performed on integers, such as addition, subtraction, and multiplication. A color, on the other hand, might be represented by three bytes denoting the amounts each of red, green, and blue, and a string representing the color's name.

Most programming languages also allow the programmer to define additional data types, usually by combining multiple elements of other types and defining the valid operations of the new data type. For example, a programmer might create a new data type named "complex number" that would include real and imaginary parts. A data type also represents a constraint placed upon the interpretation of data in a type system, describing representation, interpretation and structure of values or objects stored in computer memory. The type system uses data type information to check correctness of computer programs that access or manipulate the data.

Most data types in statistics have comparable types in computer programming, and vice versa, as shown in the following table:

Statistics	Programming
real-valued (interval scale)	floating-point
real-valued (ratio scale)	
count data (usually non-negative)	integer
binary data	Boolean
categorical data	enumerated type
random vector	list or array
random matrix	two-dimensional array
random tree	tree

Definition

(Parnas, Shore & Weiss 1976) identified five definitions of a "type" that were used—sometimes implicitly—in the literature. Types including behavior align more closely with object-oriented models, whereas a structured programming model would tend to not include code, and are called plain old data structures.

The five types are:

Syntactic
: A type is a purely syntactic label associated with a variable when it is declared. Such definitions of "type" do not give any semantic meaning to types.

Representation
: A type is defined in terms of its composition of more primitive types—often machine types.

Representation and behaviour
: A type is defined as its representation and a set of operators manipulating these representations.

Value space
: A type is a set of possible values which a variable can possess. Such definitions make it possible to speak about (disjoint) unions or Cartesian products of types.

Value space and behaviour
: A type is a set of values which a variable can possess and a set of functions that one can apply to these values.

The definition in terms of a representation was often done in imperative languages such as ALGOL and Pascal, while the definition in terms of a value space and behaviour was used in higher-level languages such as Simula and CLU.

Classes of data types

Primitive data types

Primitive data types are typically types that are built-in or basic to a language implementation.

Machine data types

All data in computers based on digital electronics is represented as bits (alternatives 0 and 1) on the lowest level. The smallest addressable unit of data is usually a group of bits called a byte (usually an octet, which is 8 bits). The unit processed by machine code instructions is called a word (as of 2011, typically 32 or 64 bits). Most instructions interpret the word as a binary number, such that a 32-bit word can represent unsigned integer values from 0 to $2^{32}-1$ or signed integer values from -2^{31} to $2^{31}-1$. Because of two's complement, the machine language and machine doesn't need to distinguish between these unsigned and signed data types for the most part.

Floating point numbers used for floating point arithmetic use a different interpretation of the bits in a word. See Floating-point arithmetic for details.

Machine data types need to be *exposed* or made available in systems or low-level programming languages, allowing fine-grained control over hardware. The C programming language, for instance, supplies integer types of various widths, such as `short` and `long`. If a corresponding native type does not exist on the target platform, the compiler will break them down into code using types that do exist. For instance, if a 32-bit integer is requested on a 16 bit platform, the compiler will tacitly treat it as an array of two 16 bit integers.

In higher level programming, machine data types are often hidden or *abstracted* as an implementation detail that would render code less portable if exposed. For instance, a generic `numeric` type might be supplied instead of integers of some specific bit-width.

Boolean type

The Boolean type represents the values true and false. Although only two values are possible, they are rarely implemented as a single binary digit for efficiency reasons. Many programming languages do not have an explicit Boolean type, instead interpreting (for instance) 0 as false and other values as true. Boolean data refers to the logical structure of how the language is interpreted to the machine language. In this case a Boolean 0 refers to the logic False. True is always a non zero, especially a one which is known as Boolean 1.

Numeric types

Such as:

- The integer data types, or "non-fractional numbers". May be sub-typed according to their ability to contain negative values (e.g. `unsigned` in C and C++). May also have a small number of predefined subtypes (such as `short` and `long` in C/C++); or allow users to freely define subranges such as 1..12 (e.g. Pascal/Ada).

- Floating point data types, usually represent values as high-precision fractional values (rational numbers, mathematically), but are sometimes misleadingly called reals (evocative of mathematical real numbers). They usually have predefined limits on both their maximum values and their precision. Typically stored internally in the form $a \times 2^b$ (where a and b are integers), but displayed in familiar decimal form.
- Fixed point data types are convenient for representing monetary values. They are often implemented internally as integers, leading to predefined limits.
- Bignum or arbitrary precision numeric types lack predefined limits. They are not primitive types, and are used sparingly for efficiency reasons.

Composite types

Composite types are derived from more than one primitive type. This can be done in a number of ways. The ways they are combined are called data structures. Composing a primitive type into a compound type generally results in a new type, e.g. *array-of-integer* is a different type to *integer*.

- An array (also called vector, list, or sequence) stores a number of elements and provide random access to individual elements. The elements of an array are typically (but not in all contexts) required to be of the same type. Arrays may be fixed-length or expandable. Indices into an array are typically required to be integers (if not, one may stress this relaxation by speaking about an associative array) from a specific range (if not all indices in that range correspond to elements, it may be a sparse array).
- Record (also called tuple or struct) Records are among the simplest data structures. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called *fields* or *members*.
- Union. A union type definition will specify which of a number of permitted primitive types may be stored in its instances, e.g. "float or long integer". Contrast with a record, which could be defined to contain a float *and* an integer; whereas, in a union, there is only one type allowed at a time.
  - A tagged union (also called a variant, variant record, discriminated union, or disjoint union) contains an additional field indicating its current type for enhanced type safety.
- A set is an abstract data structure that can store certain values, without any particular order, and no repeated values. Values themselves are not retrieved from sets, rather one tests a value for membership to obtain a boolean "in" or "not in".
- An object contains a number of data fields, like a record, and also a number of subroutines for accessing or modifying them, called methods.

Many others are possible, but they tend to be further variations and compounds of the above. For example a linked list can store the same data as an array, but provides sequential access rather than random and is built up of records in dynamic memory; though arguably a data structure rather than a type *per se*, it is also common and distinct enough that including it in a discussion of composite types can be justified.

Enumerations

The enumerated type has distinct values, which can be compared and assigned, but which do not necessarily have any particular concrete representation in the computer's memory; compilers and interpreters can represent them arbitrarily. For example, the four suits in a deck of playing cards may be four enumerators named *CLUB*, *DIAMOND*, *HEART*, *SPADE*, belonging to an enumerated type named *suit*. If a variable *V* is declared having *suit* as its data type, one can assign any of those four values to it. Some implementations allow programmers to assign integer values to the enumeration values, or even treat them as type-equivalent to integers.

String and text types

Such as:

- A character, which may be a letter of some alphabet, a digit, a blank space, a punctuation mark, etc.
- A string, which is a sequence of characters. Strings are typically used to represent words and text.

Character and string types can store sequences of characters from a character set such as ASCII. Since most character sets include the digits, it is possible to have a numeric string, such as `"1234"`. However, many languages treat these as belonging to a different type to the numeric value `1234`.

Character and string types can have different subtypes according to the required character "width". The original 7-bit wide ASCII was found to be limited, and superseded by 8 and 16-bit sets, which can encode a wide variety of non-Latin alphabets (such as Hebrew and Chinese) and other symbols. Strings may be either stretch-to-fit or of fixed size, even in the same programming language. They may also be subtyped by their maximum size.

Note: Strings are not a primitive data type in all languages. In C, for instance, they are composed from an array of characters.

Other types

Types can be based on, or derived from, the basic types explained above. In some languages, such as C, functions have a type derived from the type of their return value.

Pointers and references

The main non-composite, derived type is the pointer, a data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address. It is a primitive kind of reference. (In everyday terms, a page number in a book could be considered a piece of data that refers to another one). Pointers are often stored in a format similar to an integer; however, attempting to dereference or "look up" a pointer whose value was never a valid memory address would cause a program to crash. To ameliorate this potential problem, pointers are considered a separate type to the type of data they point to, even if the underlying representation is the same.

Function types

Abstract data types

Any type that does not specify an implementation is an abstract data type. For instance, a stack (which is an abstract type) can be implemented as an array (a contiguous block of memory containing multiple values), or as a linked list (a set of non-contiguous memory blocks linked by pointers).

Abstract types can be handled by code that does not know or "care" what underlying types are contained in them. Programming that is agnostic about concrete data types is called generic programming. Arrays and records can also contain underlying types, but are considered concrete because they specify how their contents or elements are laid out in memory.

Examples include:

- A queue is a first-in first-out list. Variations are Deque and Priority queue.
- A set can store certain values, without any particular order, and with no repeated values.
- A stack is a last-in, first out data structure.
- A tree is a hierarchical structure.
- A graph.
- A list.
- A hash, dictionary, map or associative array is a more flexible variation on a record, in which name-value pairs can be added and deleted freely.
- A smart pointer is the abstract counterpart to a pointer. Both are kinds of references.

Utility types

For convenience, high-level languages may supply ready-made "real world" data types, for instance *times*, *dates* and *monetary values* and *memory*, even where the language allows them to be built from primitive types.

Type systems

A type system associates types with computed values. By examining the flow of these values, a type system attempts to prove that no *type errors* can occur. The type system in question determines what constitutes a type error, but a type system generally seeks to guarantee that operations expecting a certain kind of value are not used with values for which that operation does not make sense.

A compiler may use the static type of a value to optimize the storage it needs and the choice of algorithms for operations on the value. In many C compilers the `float` data type, for example, is represented in 32 bits, in accord with the IEEE specification for single-precision floating point numbers. They will thus use floating-point-specific microprocessor operations on those values (floating-point addition, multiplication, etc.).

The depth of type constraints and the manner of their evaluation affect the *typing* of the language. A programming language may further associate an operation with varying concrete algorithms on each type in the case of type polymorphism. Type theory is the study of type systems, although the concrete type systems of programming languages originate from practical issues of computer architecture, compiler implementation, and language design.

Type systems may be variously static or dynamic, strong or weak typing, and so forth.

See also

- C data types
- Data dictionary
- Functional programming
- Kind
- Type theory for the mathematical models of types
- Type system for different choices in programming language typing
- Type conversion

References

1. type (http://foldoc.org/type) at the *Free On-line Dictionary of Computing*
2. Shaffer, C. A. (2011). *Data Structures & Algorithm Analysis in C++* (3rd ed.). Mineola, NY: Dover. 1.2. ISBN 978-0-486-48582-9.

Further reading

- Parnas, David L.; Shore, John E.; Weiss, David (1976). "Abstract types defined as classes of variables". *Proceedings of the 1976 Conference on Data : Abstraction, Definition and Structure*: 149–154. doi:10.1145/800237.807133 (https://doi.org/10.1145%2F800237.807133).

- Cardelli, Luca; Wegner, Peter (December 1985). "On Understanding Types, Data Abstraction, and Polymorphism" (http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf) (PDF). *ACM Computing Surveys*. **17** (4): 471–523. CiteSeerX 10.1.1.117.695 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.695). doi:10.1145/6041.6042 (https://doi.org/10.1145%2F6041.6042). ISSN 0360-0300 (https://www.worldcat.org/issn/0360-0300).
- Cleaveland, J. Craig (1986). *An Introduction to Data Types*. Addison-Wesley. ISBN 978-0201119404.

External links

- Media related to Data types at Wikimedia Commons

Retrieved from "https://en.wikipedia.org/w/index.php?title=Data_type&oldid=944671120"