Glossary

Help

Definitions of terms used in this course. For terms that have a more general meaning the definition here is as we use the term in this class.

Contents

- Signature
- String
- Stub
- Template
- Type Comment
- Value

Abstract (verb)

Abstraction means taking two or more expressions or functions that are very similar, and turning those differences into parameters of a more general-purpose **abstract function**. The Abstraction From Examples and Abstraction From Type Comments recipes cover the design of abstract functions. It is also possible to design abstract types, but we do not do that in this course.

Accumulator

An accumulator is a parameter that keeps track of information that was available earlier in a structural recursion, sometimes that information is built-up, or accumulated through the recursion. An accumulator can also be a mutable variable used with a looping construct to achieve the same effect.

Arbitrary Sized

Arbitrary Sized information (or data) is information (or data) for which the size is not known at the time the program is designed. A "player with a name and jersey number" is not arbitrary sized, it is compound data with two parts. But "all the players in the league" is arbitrary sized because we do not know ahead of time how many players there will be.

Argument

An argument is a value that passed to a function or a primitive operation when it is called. The arguments are the values that result from evaluating the operands in the function or primitive call. See parameter for a discussion of the differences between operands, arguments and parameters.

Atomic Data

Atomic data is a form of data that cannot be broken down into smaller data pieces.

Backtracking

In a backtracking search the traversal first proceeds down one branch of the tree (or graph). If that branch fails the search along that branch produces a special failure value, such as false, and the calling function then search the next branch. In this way the search backtracks to the nearest node in the tree and follows the next branch.

Boolean

`Boolean` is a primitive type that is comprised of just two values: `true` and `false`.

BST

A BST, or binary search tree, a data structure used to hold data that has been sorted in some way.

Binary means that each node has has at most two children. “Search” means that the nodes are structured in a particular sorted order: for any given node n with a key k, all of its children with a key smaller than k are in the left sub-branch. All of its children with a key larger than k are in the right sub-branch. “Tree” describes what this structure would look like if it were drawn on paper.

Closure

A closure is a locally defined function in which the body of the function uses a parameter of the enclosing function definition. The closure can be defined with local or lambda, but it must be defined inside of another function. In the example below, the helper function bigger? is a closure.

```
(define (only-bigger threshold lon)
  (local [(define (bigger? n) ;; CLOSURE (it "closes over" the value of
            (> n threshold))] ;; the threshold parameter)
    (filter bigger? lon)))
```

Compound

Compound data is a single data item that is made up of more than one related values, such as a person's first name, last name, and age. In this course we compound data is created using `define-struct`.

Constant

A constant is named value defined using `define`. It is called a constant because once it is defined it never changes.

Data

Data is the mass noun for values in our programs including numbers, strings, images, lists and compound data. In the design of programs we make a number of decisions about how to represent information as data.

Data Definition

A data definition describes a plan for representing information from the program's domain using data inside the program. A data definition includes a type comment that describes how to form the new type of data; an interpretation that describes how the data represents information in the program's domain; examples of the new type of data and a data driven template for functions that consume a single argument of the new type of data. Data definitions are designed using the How to Design Data recipe.

Directed Acyclic Graph

A directed acyclic graph (DAG) is a directed graph that does not contain cycles. In other words, it is impossible to start at a node, follow the edges of the graph and visit the same node more than once.

Directed Graph

A directed graph is a graph in which the edges between the nodes have a single direction.

Enumeration

A form of data definition in which the program domain information consists of a fixed number of

distinct values.

Expression

An expression is an element of a program that is evaluated to produce a value. See the Language page for the detailed rules for forming an expression.

Function

Functions in programs are very similar to functions in mathematics. In math, a function f(x) can be passed a value for x, and it will produce a result based on that value. Functions in programs act the same way. They have a name (in the math example this name was "f ") and one or more parameters (in the math example, the parameter was "x"). Functions also have a body, which is an expression that is evaluated to produce the resulting value of the function.

Graph

A graph is a set of nodes and a set of edges such that each edge joins two nodes.

Helper Function

In the design of a complex function it is often useful to design sub-functions that the main function can call to do part of its work. These sub-functions are sometimes called helper functions.

Image

`Image` is a primitive type of data that represents image, such as the result of a built in image function or a copy-pasted picture.

Integer

`Integer` is a primitive type of data that represents any positive or negative whole number (… -2, -1, 0, 1, 2 …).

Itemization

A form of data definition in which the data is comprised of two or more subclasses, in which at least one of the subclasses is not a distinct value.

Lambda

Lambda expressions make it possible to produce anonymous (or nameless) functions. They are convenient to use when it is necessary to pass a function as an argument to another function.

List

A list is a data structure that represents a list of items. If there is nothing in the list, its value is empty, whereas if there is data in the list (say the numbers 1 2 3), the value would be `(cons 1 (cons 2 (cons 3 empty)))`. Conses are a type of compound data: The constructor is cons, the different elements in the list can be accessed by using the selectors first and rest, and there are predicates available such as `cons?` and `empty?`

Mixed Data Itemization

A mixed data itemization is one in which at least two of the subclasses are represented by data of

different types.

Mutable Variable

A mutable variable is a variable that can have its value changed after it has been defined. Mutable variables are not used in Part 1 of the course.

Natural

`Natural` is a primitive type data that represents any non-negative whole number (0, 1, 2, 3…).

Number

`Number` is a primitive type of data that represents any number, including 0, fractions, decimal numbers and inexact numbers. For example, `1, -5, 3.4, 134.9853957` and `#i1.4142135623730951` are all `Numbers`.

Parameter

A parameter is a word used in a function declaration that represents the changing value, or the variable. It is put just after the function name so that the varying value can be specified when the function is called. When the function is called with a specific value, that value is called an argument. In the case below, `color` is a parameter, whereas `"red"` is an argument.

```
;;                /-----PARAMETER
(define (bulb colour)
  (circle 40 "solid" colour))


;;      /-----ARGUMENT
(bulb "red")
```

Predicate

A function or primitive that produces a boolean value.

Primitive

A primitive is a basic building block provided by BSL that we use when we design our programs. BSL provides primitive data and primitive operations on data.

Program Domain (aka Problem Domain)

The domain of a program is the subject matter or nature of the problem. So in a payroll system the program domain includes concepts like employees and salaries etc. In a transit system it would include concepts like bus stops and routes and schedules. The SPD approach, and in particular the HtDD and HtDW recipes stress focusing on the domain of the problem (salaries and paycheques) before the domain of the solution (`Number`, `Integer` etc.).

Purpose

A purpose is a comment that is written when a function is designed that explains in words what the function is supposed to produce. Try to keep purposes below 78 characters, but be specific!

Recursion

When a function calls itself we say that the function is recursive. When a type comment refers to itself we say that the type involves self-reference. Both are forms of recursion.

Selector

A selector is a function that is used on compound data to "select for" (or, to get the values of) the different fields of the data. The selector name consists of the data structure's name followed by a dash and then the name of the field that the selector accesses The argument for the selector should be the specific compound data for which you want to access the field. For example, with a data definition such as:

```
(define-struct cat (x y))
;; Cat is (make-cat Number Number)
;; interp: a cat on the screen, where x is the
;; x-coordinate of the cat, and y is the y-coordinate
```

Then for a cat c, the expressions would be `(cat-x c)` to get the cat's x-coordinate and `(cat-y c)` to get the cat's y-coordinate.

Signature

A signature is the first line written in a function design. It is a comment that specifies the types of arguments that the function will consume, as well as what type of data the function produces.

String

A `String` is a kind of primitive data that consists of symbols "strung" together. Strings are always enclosed in double quotation marks `"like this"`. It is important to note that if numbers are written inside strings, they are strings not numbers. `"123"` does not have the value of one hundred twenty three, since it is a `String`, whereas 123 does have the value of one hundred twenty three, since it is a Number.

Stub

A stub is a mock-version of a function that specifies the function's proper name and parameter(s), but where the body of the function is simply a value of the proper return type. See the HtD Functions page for more information on stubs.

Template

A template describes the basic structure or backbone of the function independent of its details. Data driven templates are based on the type of data the function consumes. Other kinds of templates are based on knowing something about the basic structure of the computation the function will perform. The idea of the template is to let us write down quickly what we know about the function definition "before we get to the details".

Type Comment

A type comment is a comment in a data definition that defines how the new type of data is formed.

Value

A value is a data element, such as 1, `"foo"`, `(make-cat 10 20)` etc.

Created Thu 21 Feb 2013 2:08 PM EST (UTC -0500)

Last Modified Sat 21 Sep 2013 11:09 AM EDT (UTC -0400)
