

# **HtDP: Languages as Libraries**

Version 5.1

February 14, 2011

# 1 *HtDP* Beginning Student

(require lang/htdp-beginner)

The `lang/htdp-beginner` module provides the Beginning Student language for *How to Design Programs*; see §1 “Beginning Student”.

## 2 *HtDP* Beginning Student with Abbreviations

(require lang/htdp-beginner-abbr)

The `lang/htdp-beginner-abbr` module provides the Beginning Student with Abbreviations language for *How to Design Programs*; see §2 “Beginning Student with List Abbreviations”.

### 3 *HtDP* Intermediate Student

(require lang/htdp-intermediate)

The `lang/htdp-intermediate` module provides the Intermediate Student language for *How to Design Programs*; see §3 “Intermediate Student”.

## 4 *HtDP* Intermediate Student with Lambda

(require lang/htdp-intermediate-lambda)

The `lang/htdp-intermediate-lambda` module provides the Intermediate Student with Lambda language for *How to Design Programs*; see §4 “Intermediate Student with Lambda”.

## 5 *HtDP* Advanced Student

(require lang/htdp-advanced)

The `lang/htdp-advanced` module provides the Advanced Student language for *How to Design Programs*; see §5 “Advanced Student”.

## 6 Pretty Big Text (Legacy Language)

```
(require lang/plt-pretty-big-text)
```

The `lang/plt-pretty-big-text` module is similar to the *HtDP* Advanced Student language, but with more of Racket's libraries in legacy form. It provides the bindings of `mzscheme`, `mzlib/etc`, `mzlib/file`, `mzlib/list`, `mzlib/class`, `mzlib/unit`, `mzlib/include`, `mzlib/defmacro`, `mzlib/pretty`, `mzlib/string`, `mzlib/thread`, `mzlib/math`, `mzlib/match`, `mzlib/shared`, and `lang/posn`.

## 7 Pretty Big (Legacy Language)

```
(require lang/plt-pretty-big)
```

The `lang/plt-pretty-big` module extends `lang/plt-pretty-big-text` with `scheme/gui/base` and `lang/imageeq`. This language corresponds to the Pretty Big legacy language in DrRacket.



## 8 posns in *HtDP* Languages

```
(require lang/posn)
```

---

```
(struct posn (x y)
  #:extra-constructor-name make-posn)
x : any/c
y : any/c
```

The `posn` structure type that is also provided by `lang/htdp-beginner`.

## 9 Image Equality in *HtDP* Languages

```
(require lang/imageeq)
```

---

```
(image=? i1 i2) → boolean?  
  i1 : (is-a?/c image-snip%)  
  i2 : (is-a?/c image-snip%)
```

The image-comparison operator that is also provided by `lang/htdp-beginner`.

## 10 Primitives in *HtDP* Beginner

```
(require lang/prim)
```

The `lang/prim` module several syntactic forms for use by the implementors of teachpacks, when the teachpack is to be used with the *How to Design Programs* Beginner Student languages. In Beginner Student, primitive names (for built-in procedures) are distinguished from other types of expressions, so that they can be syntactically restricted to application positions.

---

```
(define-primitive id proc-id)
```

Defines `id` to be a primitive operator whose implementation is `proc-id`, and that takes no procedures as arguments. Normally, `id` is exported from the teachpack and `proc-id` is not.

---

```
(provide-primitive id)
```

Like `define-primitive`, but the existing function `id` is exported as the primitive operator named `id`. An alternative to `define-primitive`.

---

```
(provide-primitives id ...)
```

Multiple-identifier version of `provide-primitive`.

---

```
(define-higher-order-primitive id proc-id (arg ...))
```

Defines `id` to be a primitive operator whose implementation is `proc-id`. Normally, `id` is exported from the teachpack and `proc-id` is not.

For each non-procedure argument, the corresponding `arg` should be an underscore. For each procedure argument, the corresponding `arg` should be the usual name of the procedure.

Examples:

```
(define-higher-order-primitive convert-gui convert-gui/proc (f2c))
```

---

```
(provide-higher-order-primitive id (arg ...))
```

Like `define-higher-order-primitive`, but the existing function `id` is exported as the primitive operator named `id`. An alternative to `define-higher-order-primitive`.

---

```
(first-order->higher-order expr)
```

If *expr* is an identifier for a first-order function (either a primitive or a function defined within Beginner Student), produces the function as a value; otherwise, the form is equivalent to *expr*.

This form is mainly useful for implementing syntactic forms that, like the application of a higher-order primitive, allow first-order bindings to be used in an expression position.