

# Keywords with examples

## Boo Keywords

### Index

#abstract  
#and  
#as  
#AST  
#break  
#callable  
#cast  
#char  
#class  
#constructor  
#continue  
#def  
#destructor  
#do  
#elif  
#else  
#ensure  
#enum  
#event  
#except  
#failure  
#final  
#from  
#for  
#false  
#get  
#given  
#goto  
#if  
#import  
#in  
#interface  
#internal  
#is  
#isa  
#not  
#null  
#of  
#or  
#otherwise  
#override  
#namespace  
#partial  
#pass  
#public  
#protected  
#private  
#raise  
#ref  
#retry  
#return  
#self  
#set  
#super  
#static  
#struct  
#success  
#transient  
#true  
#try  
#typeof

```
#unless
#virtual
#when
#while
#yield
```

## Keywords

### abstract

"abstract" is used to designate a class as a base class. A derivative of the abstract class must implement all of its abstract methods and properties.

#### Examples

```
abstract class base:
    pass
class foo(base):
    pass
```

### and

"and" is a logical operator that is applied to test if two boolean expressions are true.

#### Examples

```
a as bool = true
b as bool = true

if a and b:
    print "c"
```

### as

The "as" keyword declares a variables type.

#### Examples

```
intVar as int
boolVar as bool
words as string

intVar = 6
boolVar = true
words = "End of this example"
```

### AST

"AST" is used to create AST objects for use with the Boo compiler.

#### Examples

```
/*
Usage:
  result = ast: /* code block */
*/
```

## break

"break" is a keyword used to escape program execution. Typically break is used inside a loop and may be coupled with the "if" or "unless" keywords.

### Examples

```
//The break command causes the program to cease the while loop.
x = 1

while x < 10:
  print x
  x += 1  //short-hand for x = x + 1
  break
```

## callable

"callable" allows function or type to be called by another.

### Examples

```
callable Sample(param as double) as double

def Test(input as Sample):
  for i in range(0,3):
    print input(i)

Test(System.Math.Sin)
```

## cast

"cast" is a keyword used to explicitly transform a variable from one data type to another.

### Examples

```
list = List(range(5))
print list
for item in list:
  print ((item cast decimal) / 3)
print '---'
for item as int in list:
  print item * item
```

## char

"char" is a data type representing a single character. The char data type is distinct from a string containing a single character. char('t') refers to a System.Char type, whereas "t" or 't' is a System.String.

## Examples

```
a1 = char('a')
print a1
```

## class

"class" is a definition of an object including its properties and methods.

## Examples

```
class Foo():
    pass

class Dog():
    [Property (Name)] _name as string

    def constructor(name as string):
        Name = name

    def Bark():
        print (Name + " says woof")

fido = Dog ("Fido")
fido.Bark()
```

## constructor

"constructor" is a method belonging to a class that is used to define how an instance of the class should be created. The constructor may include input parameters and may be overloaded.

## Examples

*see the examples for the keyword "class"*

## continue

"continue" is a keyword used to resume program execution at the end of the current loop.

The continue keyword is used when looping. It will cause the position of the code to return to the start of the loop (as long as the condition still holds).

## Examples

```
for i in range(10):
    continue if i % 2 == 0
    print i
```

## def

"def" is used to define a new function or method.

## Examples

```
def intReflect(a as int):
    intValue as int
    intValue = a
    return intValue

def printFoo():
    print "Foo"
```

## destructor

"destructor" is used to destroy objects. Destructors are necessary to release memory used by non-managed resources in the .NET CLI. Destructors should never be called explicitly. They can be invoked by implementing the IDisposable() interface.

## Examples

```
class Dog:
    [Property (Name)] _name as string

    def constructor():
        Name = 'Fido'

    def destructor():
        print "$Name is no more"
```

## do

"do" is synonymous with 'def' for closures. However, "do" reads as an imperative and therefore should be used in an active sense.

## Examples

### Example 1

```
list.forEach() do(item):
    print item

###and def for use in definition like:
output = def(str as string):
    print str
```

### Example 2

```
c = do(x as int):
    pass
### lines above are the same as...
c = def(x as int):
    pass
```

## elif

"elif" is similar the same as the "if" conditional statement in form, except that it needs to be preceded by an if statement or another elif statement and that it is only evaluated (checked) if the if/elif statement preceding it evaluates to false.

If one of the preceding if/elifs statements evaluates to true, the rest of the elifs will not be evaluated, thus sparing extra CPU power from a pointless task.

### Examples

```
x = 3

if x == 1:
    print "One."
elif x == 2:
    print "Two."
elif x == 3:
    print "Three."
```

## else

"else" is defines a statement that will be executed should a preceding "if" condition fail.

### Examples

```
x = 666
//Block form.
if x > 0: //Evaluates to true
    print "x is greater than 0; specifically, x is $x"
else:
    //Executes when the "if" above evaluates to false
    print "x is not greater than than 0; specifically, x is $x"
```

## ensure

"ensure" is used with the "try" and "except" keywords to guarantee a certain block of code runs whether the try/except block is successful or not. "ensure" is often used to add some post executions to an exception event.

### Examples

```

import System

class Dog:
    [Property (Name)] _name as string
    def constructor():
        Name = 'Fido'
    def Bark():
        print "woof woof"

### The name will match so we will bark once. The ensure keyword reminds us to place
the dog in the pen and wait for keyboard input to continue.
try:
    fido = Dog()
    if fido.Name == 'Fido':
        fido.Bark()
    else:
        fido.Bark()
        fido.Bark()
        raise "throw an exception"
except e:
    print("The dog barks too much. error: " + e)
ensure:
    print("Always put the dog back in the pen.")
    Console.ReadLine()

### The name does not match we bark twice and report an exception. Again, the ensure
statement executes and reminds us to place the dog back into the pen.
try:
    fido = Dog()
    if fido.Name == 'fluffy':
        fido.Bark()
    else:
        fido.Bark()
        fido.Bark()
        raise "throw an exception"
except e:
    print("The dog barks too much. error: " + e)
ensure:
    print("Always put the dog back in the pen.")
    Console.ReadLine()

```

## enum

"enum" is used to create a list of static values. Internally the names are assigned to an Int32 value.

## Examples

```
enum WeekDays :
    Mon
    Tue
    Wed
    Thu
    Fri

print join(WeekDays.GetNames(WeekDays))
print WeekDays.GetName(WeekDays, 2)
print WeekDays.Tue.GetHashCode()
print WeekDays.Mon.GetType()
```

## event

"event" is (insert text here)

## except

"except" is keyword use to identify a block of code that is to be executed if the "try" block fails.

## Examples

See examples under the "ensure" keyword.

## failure

"failure" is not yet implemented in boo.

## final

"final" is a keyword used to identify a class that cannot have subclasses. final may also be used to declare a field as a constant.

## Examples

### Example 1

```
final class Rectangle():
    pass

### Boo will complain that Rectangle cannot be extended
class Square(Rectangle):
    pass
```

### Example 2

```
class C:
    final A = 2
    final B as string //may be declared once in the constructor

    static final zed = 3 //same as C# const keyword
```



## from

"from" is used with the "import" keyword to identify the assembly being imported from. Form usage is "import TARGET (from ASSEMBLY). The "from" keyword is optional.

### Examples

```
import Gtk from "gtk-sharp"  
import System.Drawing  
  
Application.Init()
```

## for

"for" is used to loop through items in a series. "for" loops are frequently used with a range or a listarray.

### Examples

```
flock = ['cardinal', 'flamingo', 'hummingbird']  
for bird in flock:  
    print bird  
  
for i in range(3):  
    print i
```

## false

"false" represents a negative boolean outcome.

### Examples

```
j as bool = false  
if j == false:  
    print "j is false."
```

## get

"get" is used to identify a field that is exposed for external access. Use "get" to make a field available as read-only. Use "set" to add write access. "get" is suffixed by a colon when implemented and includes a return statement. It is possible to modify the value of the field being returned. See example 1.

"get" is also used when defining an interface to define which fields should be implemented as accessible. When "get" is used to define an interface the colon and return statements are excluded. See example 2.

### Examples

#### Example 1

```

import System

class Person:
    _fname as string
    FirstName as string:
        get:
            return "Master " + _fname

    def constructor(fname, lname):
        raise ArgumentException("fname") if fname is null
        _fname = fname

jax = Person("jax", "lax")
print jax.FirstName
#print jax._fname      #Inaccessible due to its protection level

```

## Example 2

```

interface IAnimal:
    Name as string:
        get

class Dog(IAnimal):
    Name:
        get:
            return "charlie"

chuck = Dog()
print chuck.Name

```

## given

"given" is used as the entry to a "given ... when" loop. "given" identifies a state. A series of "when" statements may be executed based on the identified state. \_ The "given" keyword is currently not implemented. \_

## Examples

### Example 1

```

lista = ['uno', 2, 'tres', 4, false]
for i in lista:
    given i:
        when string:
            print i
        when int:
            print i*2
        otherwise:
            print "no condition met"

```

### Example 2

```
given c:
  when isa Car:
    c.Drive()
  when isa Plane:
    c.Fly()
  when isa string:
    given c:
      when ~= "some text":
        print "nested given on a string"
      otherwise:
        print "otherwise is similar to else"
    otherwise:
      print "I can't operate this $c"
```

## goto

"goto" exits a line of code and moves to a named line in the code. The named line must be prefixed with a colon. Good programming practice eschews the use of "goto"

The example below names two lines ":start" and "test". They are referenced in the code by separate goto statements. This example produces an endless loop. The "ensure" statement includes a Console.ReadLine() that prevents the loop from continuing without user input.

## Examples

```
i as int = 0

:start
print "ding"
i += 1
goto start if (i<3)
:test
print "a test"

try:
  print "stuff"
  goto test
except e:
  print "The dog barks too much. error: " + e
ensure:
  print "Always put the dog back in the pen."
  System.Console.ReadLine()
```

## if

"if" is a conditional statement, followed by a statement that either evaluates to true or false. In block form, the code within the block is executed only if the expression following the if evaluates to true.

The if statement can be used to selectively execute a line of code by placing "if <expression>" at the very end of the statement. This form of the if conditional is useful in circumstances when you are only going to perform one operation based entirely on an expression: this makes the code cleaner to read than an unnecessary if block.

## Examples

```
x = 666
//Block form.
if x > 0: //Evaluates to true
    print "x is greater than 0; specifically, x is $x"

//Selectively execute a line of code.
print "x is greater than 0; specifically, x is $x" if x > 0 //Equivalent of the above.
```

## import

"import" is used to include a namespace from other assemblies within your program. If the assembly is not automatically included, the "from" keyword must be included to identify the respective assembly.

### Example

#### Example 1

```
import System
import Gtk from "gtk-sharp"

###prints 3.1415926535879
print Math.PI

###prints an Error
print PI
```

#### Example 2

```
import System.Math

### prints 3.1415926535879
print PI
```

## in

"in" is used in conjunction with "for" to iterate through items in a list. "in" may also be used to test items in a set.

### Examples

#### Example 1

See examples for the keyword "for".

#### Example 2

```
aList = [1,2,3]
if 1 in aList:
    print "there is a one in there"
```

## interface

"interface" is used to define the fields and methods that may be implemented by a class. The implementation is never performed by the interface. Interfaces allow you to establish an API that is the basis for other classes.

## Examples

```
interface IAnimal:
    Name as string:
        get

class Dog(IAnimal):
    Name:
        get:
            return "charlie"

chuck = Dog()
print chuck.Name
```

## internal

"internal" is a keyword that precedes a class definition to limit the class to the assembly in which it is found.

## Examples

```
internal class Cat:
    pass
```

## is

"is" is an equivalence operator keyword that is used to test a value. "is" may not be used with ints, doubles, or boolean types. "is" is commonly used to test for null.

## Examples

### Example 1

```
lol = null
print lol is null
print lol is not null
```

### Example 2

```
class a():
    pass

b = a()
c = a()

print b is c //false
print b is a //false

d = a
print d is a //true
```

## isa

"isa" determines if one element is an instance of a specific type.

### Examples

#### Example 1

```
class A:
    pass
class B(A):
    pass
class C(B):
    pass
print C() isa A #true
print C() isa B #true
print B() isa C #false
```

#### Example 2

```
class Cat:
    pass

dale = Cat()

if dale isa Cat:
    print "dale really isa cat"

j as string = "the jig is up"
if j isa string:
    print "j really isa string"
```

## not

"not" is used with "is" to perform a negative comparison. "not" can also be used in logical expressions.

### Examples

#### Example 1

```
class Cat:
    pass
class Dog:
    pass

dale = Cat()
if not dale isa Dog:
    print "dale must be a cat"
```

## Example 2

```
i = 0
if not i == 1:
    print "i is not one"
```

## null

"null" is a keyword used to specify a value is absent.

## Examples

```
j as string = null

if j is null:
    print "j is null. Assign a value"
```

## of

"of" is used to specify type arguments to a generic type or method.

## Examples

```
myList = List[of int]()
myList.Add(1) # success!
myList.Add('f') # failure, oh horrible, horrible failure.
```

```
import System
names = ("Betty", "Charlie", "Allison")
Array.Sort[of string](names)
```

## or

"or" is a logical operator that is applied to test if either of two boolean expressions are true.

## Examples

```
a as bool = true
b as bool = false

if a or b:
    print "c"
```

## otherwise

"otherwise" is part of the conditional phrase "given ... when ... otherwise". The otherwise block is executed for a given state if none of the when conditions match. `_` The otherwise keyword is not yet implemented `_`

## Examples

See examples for "given".

## override

"override" is used in a derived class to declare that a method is to be used instead of the inherited method. "override" may only be used on methods that are defined as "virtual" or "abstract" in the parent class.

## Examples

```
class Base:
    virtual def Execute():
        print 'From Base'

class Derived(Base):
    override def Execute():
        print 'From Derived'

b = Base()
d = Derived()
b.Execute()
d.Execute()
(d cast Base).Execute()
```

## namespace

"namespace" is a name that uniquely identifies a set of objects so there is no ambiguity when objects from different sources are used together. To declare a namespace place the namespace followed by the name you choose at the top of the file.

## Examples

```
_need an example added here_
```

## partial

"partial" is (insert text here)

## pass

"pass" is a keyword used when you do not want to do anything in a block of code.



## Examples

### Example 1

```
def Cat():  
    pass
```

### Example 2

```
if x is true:  
    pass
```

## public

"public" is used to define a class, method, or field as available to all. "public class" is never required because a defined class defaults to public.

### Example

#### Example 1

```
public class Cat:  
    pass
```

#### Example 2

```
public def simpleFunction():  
    return 1
```

## protected

"protected" is a keyword used to declare a class, method, or field as visible only within its containing class. Fields are by default protected. Prefixing a field name with an underscore is recommended practice.

## Examples

### Example 1

```
class Cat:  
    _name as string
```

### Example 2

```
class Dog:  
    protected def digest():  
        pass
```

## private

"private" is keyword used to declare a class, method, or field visible within only its containing class and inherited classes..

### Examples

#### Example 1

```
private class Dog:  
    pass
```

#### Example 2

```
class Human:  
    private def DigestFood():  
        pass
```

#### Example 3

```
class Human:  
    private heart as string
```

## raise

"raise" is (insert text here)

## ref

"ref" makes a parameter be passed by reference instead of by value. This allows you to change a variable's value outside of the context where it is being used

### Examples

```
def dobyref(ref x as int):  
    x = 4  
  
x = 1  
print x //-->1  
dobyref(x)  
print x //-->4
```

## retry

"retry" is not yet implemented.

## return

"return" is a keyword use to state the value to be returned from a function definition

### Examples

## Example 1

```
def printOne():  
    return "One"
```

## Example 2

```
def Add(intA as int, intB as int):  
    return (intA + intB)
```

## self

"self" is used to reference the current class. "self" is not required for boo but may be used to add clarity to the code. "self" is synonymous with the c# keyword "this".

## Examples

```
class Point():  
    [property(Xcoordinate)] _xcoordinate as double  
    [property(Ycoordinate)] _ycoordinate as double  
    def constructor():  
        pass  
    def constructor(one as double, two as double):  
        self.Xcoordinate = one  
        self.Ycoordinate = two
```

## set

"set" is a keyword used to define a field as writeable.

## Examples

```
class Cat:  
    _name as string  
    Name as string:  
    get:  
        return _name  
    set:  
        _name = value  
  
fluffy = Cat()  
fluffy.Name = 'Fluffy'
```

## static

"static" is (insert text here)

## struct

"struct" is short for structure. A structure is similar to a class except it defines value types rather than reference types.

Refer to the Boo Primer for more information on structures.

## Examples

```
struct Coordinate:
    X as int
    Y as int
    def constructor(x as int, y as int):
        X = x
        Y = y

c as Coordinate
print c.X, c.Y
c = Coordinate(3, 5)
print c.X, c.Y
```

## success

"success" is not yet implemented.

## super

"super" is used to reference a base class from a child class when one wants to execute the base behavior.

## Examples

```
class SuperClass:
    def printMethod():
        print "Printed in SuperClass"

class SubClass(SuperClass):
    def printMethod():
        super.printMethod()
        print "Printed in SubClass"

s = SubClass()
s.printMethod()
```

## transient

"transient" transient marks a member as not to be serialized. By default, all members in Boo are serializable.

## Examples

```
_please insert example_
```

## true

"true" is keyword used to represent a positive boolean outcome.

## Examples

```
a as bool = true

if a is true:
    print "as true as true can be"
```

## try

"try" is used with the "ensure" and "except" keywords to test whether a block of code executes without error.

## Examples

```
_see keyword ensure for examples_
```

## typeof

typeof returns a Type instance. Unnecessary, in Boo since you can pass by type directly.

## Examples

```
anInteger = typeof(int)
#or, the boo way:
anotherInteger = int
```

## unless

"unless" is similar to the "if" statement, except that it executes the block of code **unless** the expression is true.

## Examples

```
x = 0
unless x >= 54:
    print "x is less than 54."
```

## virtual

"virtual" is a keyword that may precede the 'def' keyword when the developer wishes to provide the ability to override a defined method in a child class. The 'virtual' keyword is used in the parent class.

## Examples

```
class Mammal:
    virtual def MakeSound():
        print "Roar"

class Dog(Mammal):
    override def MakeSound():
        print "Bark"
```

## when

"when" is used with the "given" keyword to identify the condition in a which the "given" value may be executed. \_b "when" is currently not implemented.

## Examples

see examples for the "given" keyword.

## while

"while" will execute a block of code as long as the expression it evaluates is true.

It is useful in cases where a variable must constantly be evaluated (in another thread, perhaps) , such as checking to make sure a socket still has a connection before emptying a buffer (filled by another thread, perhaps).

## Examples

```
i = 0

while i > 3:
    print i
```

## yield

"yield" is similar to "return" only it can be called multiple times within a single method.

## Examples

```
def TestGenerator():
    i = 1
    yield i
    for x in range(10):
        i *= 2
        yield i

print List(TestGenerator())
```