Integer Operators

A

This documentation was generated from the Python documentation available by typing *help*(*int*) in the Python shell. In this documentation the variables *x*, *y*, and *z* refer to integers.

Operator	Returns	Comments
x+y	int	Returns the sum of x and y.
x-y	int	Returns the difference of x and y.
x*y	int	Returns the product of x and y.
x/y	float	Returns the quotient of x divided by y.
x//y	int	Returns the integer quotient of x divided by y.
x%y	int	Returns x modulo y. This is the remainder of dividing x by y.
-x	int	Returns the negation of x.
x&y	int	Returns the bit-wise *and* of x and y.
x | y	int	Returns the bit-wise *or* of x and y.
x^y	int	Returns the bit-wise *exclusive or* of x and y.
x<<y	int	Returns a bit-wise shift left of x by y bits. Shifting left by 1 bit multiplies x by 2.
x>>y	int	Returns a bit-wise right shift of x by y bits.
~x	int	Returns an integer where each bit in the x has been inverted. $x + \ x = -1$ for all x.
abs(x)	int	Returns the absolute value of x.
divmod(x, y)	(q,r)	Returns the quotient q and the remainder r as a tuple.
float(x)	float	Returns the float representation of x.
hex(x)	str	Returns a hexadecimal representation of x as a string.
int(x)	int	Returns x.
oct(x)	str	Return an octal representation of x as a string.
pow(x, y[, z])	int	Returns x to the y power modulo z. If z is not specified then it returns x to the y power.
repr(x)	str	Returns a string representation of x.
str(x)	str	Returns a string representation of x.

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

Float Operators

B

This documentation was generated from the Python documentation available by typing *help*(*float*) in the Python shell. In this documentation at least one of the variables *x* and *y* refer to floats.

Operator	Returns	Comments
x+y	float	Returns the sum of x and y.
x-y	float	Returns the difference of x and y.
x*y	float	Returns the product of x and y.
x/y	float	Returns the quotient of x divided by y.
x//y	float	Returns the quotient of integer division of x divided by y. However, the result is still a float.
x%y	float	Returns x modulo y. This is the remainder of dividing x by y.
abs(x)	int	Returns the absolute value of x.
divmod(x, y)	(q,r)	Returns the quotient q and the remainder r as a tuple. Both q and r are floats, but integer division is performed. The value r is the whole and fractional part of any remainder. The value q is a whole number.
float(x)	float	Returns the float representation of x.
int(x)	int	Returns the floor of x as an integer.
pow(x, y)	float	Returns x to the y power.
repr(x)	str	Returns a string representation of x.
str(x)	str	Returns a string representation of x.

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

String Operators and Methods

C

This documentation was generated from the Python documentation available by typing *help*(*str*) in the Python shell. In the documentation found here the variables *s* and *t* are references to strings.

Operator	Returns	Comments
s+t	str	Return a new string which is the concatenation of s and t.
s in t	bool	Returns True if s is a substring of t and False otherwise.
s==t	bool	Returns True if s and t refer to strings with the same sequence of characters.
s>=t	bool	Returns True if s is lexicographically greater than or equal to t.
s<=t	bool	Returns True if s is lexicographically less than or equal to t.
s>t	bool	Returns True if s is lexicographically greater than t.
s<t	bool	Returns True if s is lexicographically less than t.
s!=t	bool	Returns True if s is lexicographically not equal to t.
s[i]	str	Returns the character at index i in the string. If i is negative then it returns the character at index len(s)-i.
s[[i]:[j]]	str	Returns the slice of characters starting at index i and extending to index j-1 in the string. If i is omitted then the slice begins at index 0. If j is omitted then the slice extends to the end of the list. If i is negative then it returns the slice starting at index len(s)+i (and likewise for the slice ending at j).
s * i	str	Returns a new string with s repeated i times.
i * s	str	Returns a new string with s repeated i times.
chr(i)	str	Return the ASCII character equivalent of the integer i.
float(s)	float	Returns the float contained in the string s.
int(s)	int	Returns the integer contained in the string s.
len(s)	int	Returns the number of characters in s.
ord(s)	int	Returns the ASCII decimal equivalent of the single character string s.
repr(s)		Returns a string representation of s. This adds an extra pair of quotes to s.
str(s)	str	Returns a string representation of s. In this case you get just the string s.

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

Method	Returns	Comments
s.capitalize()	str	Returns a copy of the string s with the first character upper case.
s.center(width[, fillchar])	str	Returns s centered in a string of length width. Padding is done using the specified fill character (default is a space)
s.count(sub[, start[, end]])	int	Returns the number of non-overlapping occurrences of substring sub in string s[start:end]. Optional arguments start and end are interpreted as in slice notation.
s.encode([encoding[, errors]])	bytes	Encodes s using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.
s.endswith(suffix[, start[, end]])	bool	Returns True if s ends with the specified suffix, False otherwise. With optional start, test s beginning at that position. With optional end, stop comparing s at that position. suffix can also be a tuple of strings to try.
s.expandtabs([tabsize])	str	Returns a copy of s where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.
s.find(sub[, start[, end]])	int	Returns the lowest index in s where substring sub is found, such that sub is contained within s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.
s.format(*args, **kwargs)	str	
s.index(sub[, start[, end]])	int	Like s.find() but raise ValueError when the substring is not found.
s.isalnum()	bool	Returns True if all characters in s are alphanumeric and there is at least one character in s, False otherwise.
s.isalpha()	bool	Returns True if all characters in s are alphabetic and there is at least one character in s, False otherwise.
s.isdecimal()	bool	Returns True if there are only decimal characters in s, False otherwise.
s.isdigit()	bool	Returns True if all characters in s are digits and there is at least one character in s, False otherwise.
s.isidentifier()	bool	Returns True if s is a valid identifier according to the language definition.

Method	Returns	Comments
s.islower()	bool	Returns True if all cased characters in s are lowercase and there is at least one cased character in s, False otherwise.
s.isnumeric()	bool	Returns True if there are only numeric characters in s, False otherwise.
s.isprintable()	bool	Returns True if all characters in s are considered printable in repr() or s is empty, False otherwise.
s.isspace()	bool	Returns True if all characters in s are whitespace and there is at least one character in s, False otherwise.
s.istitle()	bool	Returns True if s is a titlecased string and there is at least one character in s, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.
s.isupper()	bool	Returns True if all cased characters in s are uppercase and there is at least one cased character in s, False otherwise.
s.join(sequence)	str	Returns a string which is the concatenation of the strings in the sequence. The separator between elements is s.
s.ljust(width[, fillchar])	str	Returns s left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).
s.lower()	str	Returns a copy of the string s converted to lowercase.
s.lstrip([chars])	str	Returns a copy of the string s with leading whitespace removed. If chars is given and not None, remove characters in chars instead.
s.partition(sep)	(h,sep,t)	Searches for the separator sep in s, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns s and two empty strings.
s.replace (old, new[, count])	str	Returns a copy of s with all occurrences of substring old replaced by new. If the optional argument count is given, only the first count occurrences are replaced.
s.rfind(sub[, start[, end]])	int	Returns the highest index in s where substring sub is found, such that sub is contained within s[start:end]. Optional arguments start and end are interpreted as in slice notation. Returns -1 on failure.
s.rindex(sub[, start[, end]])	int	Like s.rfind() but raise ValueError when the substring is not found.
s.rjust(width[, fillchar])	str	Returns s right-justified in a string of length width. Padding is done using the specified fill character (default is a space).

Method	Returns	Comments
s.rpartition(sep)	(t,sep,h)	Searches for the separator sep in s, starting at the end of s, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns two empty strings and s.
s.rsplit([sep[, maxsplit]])	string list	Returns a list of the words in s, using sep as the delimiter string, starting at the end of the string and working to the front. If maxsplit is given, at most maxsplit splits are done. If sep is not specified, any whitespace string is a separator.
s.rstrip([chars])	str	Returns a copy of the string s with trailing whitespace removed. If chars is given and not None, removes characters in chars instead.
s.split([sep[, maxsplit]])	string list	Returns a list of the words in s, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done. If sep is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.
s.splitlines([keepends])	string list	Returns a list of the lines in s, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.
s.startswith(prefix[, start[, end]])	bool	Returns True if s starts with the specified prefix, False otherwise. With optional start, test s beginning at that position. With optional end, stop comparing s at that position. prefix can also be a tuple of strings to try.
s.strip([chars])	str	Returns a copy of the string s with leading and trailing whitespace removed. If chars is given and not None, removes characters in chars instead.
s.swapcase()	str	Returns a copy of s with uppercase characters converted to lowercase and vice versa.
s.title()	str	Returns a titlecased version of s, i.e. words start with title case characters, all remaining cased characters have lower case.
s.translate(table)	str	Returns a copy of the string s, where all characters have been mapped through the given translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None. Unmapped characters are left untouched. Characters mapped to None are deleted.
s.upper()	str	Returns a copy of s converted to uppercase.
s.zfill(width)	str	Pad a numeric string s with zeros on the left, to fill a field of the specified width. The string s is never truncated.

List Operators and Methods

D

This documentation was generated from the Python documentation available by typing *help*(*list*) in the Python shell. In the documentation found here the variables *x* and *y* are references to lists.

Method	Returns	Comments
list()	list	Returns a new empty list. You can also use `[]` to initialize a new empty list.
list(sequence)	list	Returns new list initialized from sequence's items.
[item [,item]+ **]**	list	Writing a number of comma-separated items in square brackets constructs a new list of those items.
x+y	list	Returns a new list containing the concatenation of the items in x and y.
e in x	bool	Returns True if the item e is in x and False otherwise.
del x[i]		Deletes the item at index i in x. This is not an expression and does not return a value.
x==y	bool	Returns True if x and y contain the same number of items and each of those corresponding items are pairwise equal.
x>=y	bool	Returns True if x is greater than or equal to y according to a lexicographical ordering of the elements in x and y. If x and y have different lengths their items are == up to the shortest length, then this returns True if x is longer than y.
x<=y	bool	Returns True if x is lexicographically before y or equal to y and False otherwise.
x>y	bool	Returns True if x is lexicographically after y and False otherwise.
x<y	bool	Returns True if x is lexicographically before y and False otherwise.
x!=y	bool	Returns True if x and y are of different length or if some item of x is not == to some item of y. Otherwise it returns False.
x[i]	item	Returns the item at index i of x.

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

Method	Returns	Comments
x[[i]:[j]]	list	Returns the slice of items starting at index i and extending to index j-1 in the string. If i is omitted then the slice begins at index 0. If j is omitted then the slice extends to the end of the list. If i is negative then it returns the slice starting at index len(x)+i (and likewise for the slice ending at j).
x[i]=e		Assigns the position at index i the value of e in x. The list x must already have an item at index i before this assignment occurs. In other words, assigning an item to a list in this way will not extend the length of the list to accommodate it.
x+=y		This mutates the list x to append the items in y.
x*=i		This mutates the list x to be i copies of the original x.
iter(x)	iterator	Returns an iterator over x.
len(x)	int	Returns the number of items in x.
x*i	list	Returns a new list with the items of x repeated i times.
i*x	list	Returns a new list with the items of x repeated i times.
repr(x)	str	Returns a string representation of x.
x.append(e)	None	This mutates the value of x to add e as its last element. The function returns None, but the return value is irrelevant since it mutates x.
x.count(e)	int	Returns the number of occurrences of e in x by using == equality.
x.extend(iter)	None	Mutates x by appending elements from the iterable, iter.
x.index(e,[i,[j]])	int	Returns the first index of an element that == e between the start index, i, and the stop index, j-1. It raises ValueError if the value is not present in the specified sequence. If j is omitted then it searches to the end of the list. If i is omitted then it searches from the beginning of the list.
x.insert(i, e)	None	Insert e before index i in x, mutating x.
x.pop([index])	item	Remove and return the item at index. If index is omitted then the item at len(x)-1 is removed. The pop method returns the item and mutates x. It raises IndexError if list is empty or index is out of range.
x.remove(e)	None	remove first occurrence of e in x, mutating x. It raises ValueError if the value is not present.
x.reverse()	None	Reverses all the items in x, mutating x.
x.sort()	None	Sorts all the items of x according to their natural ordering as determined by the item's __cmp__ method, mutating x. Two keyword parameters are possible: key and reverse. If reverse=True is specified, then the result of sorting will have the list in reverse of the natural ordering. If key=f is specified then f must be a function that takes an item of x and returns the value of that item that should be used as the key when sorting.

Dictionary Operators and Methods

E

This documentation was generated from the Python documentation available by typing *help*(*dict*) in the Python shell. In the documentation found here the variable *D* is a reference to a dictionary. A few methods were omitted here for brevity.

Method	Returns	Comments
dict()	dict	new empty dictionary.
dict(mapping)	dict	new dictionary initialized from a mapping object's (key, value) pairs.
dict(seq)	dict	new dictionary initialized as if via: D = {} for k, v in seq: D[k] = v
dict(**kwargs)	dict	new dictionary initialized with the name=value pairs in the keyword arg list. For example: dict(one=1, two=2)
k in D	bool	True if D has key k, else False
del D[k]		Deletes key k from dictionary D.
D1==D2	bool	Returns True if dictionaries D1 and D2 have same keys mapped to same values.
D[k]	value type	Returns value k maps to in D. If k is not mapped, it raises a KeyError exception.
iter(D)	iterator	Returns an iterator over D.
len(D)	int	Returns the number of keys in D.
D1!=D2	bool	Returns True if D1 and D2 have any different keys or keys map to different values.
repr(D)	str	Returns a string representation of D.
D[k]=e		Stores the key,value pair k,e in D.
D.clear()	None	Remove all items from D.
D.copy()	dict	a shallow copy of D
D.get(k[,e])	value type	D[k] if k in D, else e. e defaults to None.
D.items()	items	a set-like object providing a view on D's items

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

Method	Returns	Comments
D.keys()	keys	a set-like object providing a view on D's keys
D.pop(k[,e])	v	remove specified key and return the corresponding value. If key is not found, e is returned if given, otherwise KeyError is raised
D.popitem()	(k, v)	remove and return some (key, value) pair as a 2-tuple; but raise KeyError if D is empty.
D.setdefault(k[,e])	D.get(k,e)	Returns D.get(k,e) and also sets d[k]=e if k not in D
D.update(E, **F)	None	Update D from dict/iterable E and F. If E has a .keys() method, does: for k in E: D[k] = E[k] If E lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
D.values()	values	an object providing a view on D's values

Turtle Methods

F

This documentation was generated from the Python documentation available by typing

```
from turtle import *
help(Turtle)
```

in the Python shell. In the documentation found here the variable *turtle* is a reference to a Turtle object. This is a subset of that documentation. To see complete documentation use the Python help system as described above.

Method	Description

turtle.back(distance)	Aliases: backward bk
	
	Argument:
	distance – a number
	
	Move the turtle backward by distance, opposite to the direction the turtle is headed. Do not change the turtle's heading.
	
	Example (for a Turtle instance named turtle):
	>>> turtle.position()
	(0.00, 0.00)
	>>> turtle.backward(30)
	>>> turtle.position()
	(-30.00, 0.00)

turtle.begin_fill()	Called just before drawing a shape to be filled.
	
	Example (for a Turtle instance named turtle):
	>>> turtle.color("black", "red")
	>>> turtle.begin_fill()
	>>> turtle.circle(60)
	>>> turtle.end_fill()

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science, 205
DOI 10.1007/978-1-84996-537-8, © Springer-Verlag London Limited 2011

F

Method Description
turtle.begin_poly() Start recording the vertices of a polygon. Current turtle position is first point of polygon. Example (for a Turtle instance named turtle): >>> turtle.begin_poly()

turtle.circle(radius, extent=None, steps=None)
Arguments:
radius—a number
extent (optional)—a number
steps (optional)—an integer
 	
Draw a circle with given radius. The center is radius units left of the turtle; extent—an angle—determines which part of the circle is drawn. If extent is not given, draw the entire circle. If extent is not a full circle, one endpoint of the arc is the current pen position. Draw the arc in counterclockwise direction if radius is positive, otherwise in clockwise direction. Finally the direction of the turtle is changed by the amount of extent.
 	
As the circle is approximated by an inscribed regular polygon, steps determines the number of steps to use. If not given, it will be calculated automatically. Maybe used to draw regular polygons.
 	
call: circle(radius) # full circle
–or: circle(radius, extent) # arc
–or: circle(radius, extent, steps)
–or: circle(radius, steps=6) # 6-sided polygon
 	
Example (for a Turtle instance named turtle):
>>> turtle.circle(50)
>>> turtle.circle(120, 180) # semicircle

turtle.clear() Delete the turtle's drawings from the screen. Do not move turtle. State and position of the turtle as well as drawings of other turtles are not affected. Examples (for a Turtle instance named turtle): >>> turtle.clear()
turtle.color(*args) Arguments: Several input formats are allowed. They use 0, 1, 2, or 3 arguments as follows: color() Return the current pencolor and the current fillcolor as a pair of color specification strings as are returned by pencolor and fillcolor.

Method	Description

	color(colorstring), color((r,g,b)), color(r,g,b)
	inputs as in pencolor, set both, fillcolor and pencolor,
	to the given value.
	color(colorstring1, colorstring2),
	color((r1,g1,b1), (r2,g2,b2))
	equivalent to pencolor(colorstring1) and fillcolor(colorstring2)
	and analogously, if the other input format is used.
	If turtleshape is a polygon, outline and interior of that polygon
	is drawn with the newly set colors.
	For mor info see: pencolor, fillcolor
	
	Example (for a Turtle instance named turtle):
	>>> turtle.color('red', 'green')
	>>> turtle.color()
	('red', 'green')
	>>> colormode(255)
	>>> color((40, 80, 120), (160, 200, 240))
	>>> color()
	('#285078', '#a0c8f0')

turtle.degrees()	Set the angle measurement units to degrees.
	
	Example (for a Turtle instance named turtle):
	>>> turtle.heading()
	1.5707963267948966
	>>> turtle.degrees()
	>>> turtle.heading()
	90.0

turtle.dot(size=None, *color)	Optional arguments:
	size—an integer >= 1 (if given)
	color—a colorstring or a numeric color tuple
	
	Draw a circular dot with diameter size, using color.
	If size is not given, the maximum of pensize+4 and 2*pensize is used.
	
	Example (for a Turtle instance named turtle):
	>>> turtle.dot()
	>>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)

turtle.end_fill()	Fill the shape drawn after the call begin_fill().
	
	Example (for a Turtle instance named turtle):
	>>> turtle.color("black", "red")
	>>> turtle.begin_fill()
	>>> turtle.circle(60)
	>>> turtle.end_fill()

F

Method Description
turtle.end_poly() Stop recording the vertices of a polygon. Current turtle position is last point of polygon. This will be connected with the first point. Example (for a Turtle instance named turtle): >>> turtle.end_poly()

turtle.filling()
Return fillstate (True if filling, False else).
 	
Example (for a Turtle instance named turtle):
>>> turtle.begin_fill()
>>> if turtle.filling():
turtle.pensize(5)
else:
turtle.pensize(3)

turtle.fillcolor(*args)
Return or set the fillcolor.
 	
Arguments:
Four input formats are allowed:
– fillcolor()
Return the current fillcolor as color specification string,
possibly in hex-number format (see example).
May be used as input to another color/pencolor/fillcolor call.
– fillcolor(colorstring)
s is a Tk color specification string, such as “red” or “yellow”
– fillcolor((r, g, b))
a tuple of r, g, and b, which represent, an RGB color,
and each of r, g, and b are in the range 0..colormode,
where colormode is either 1.0 or 255
– fillcolor(r, g, b)
r, g, and b represent an RGB color, and each of r, g, and b
are in the range 0..colormode
 	
If turtleshape is a polygon, the interior of that polygon is drawn
with the newly set fillcolor.
 	
Example (for a Turtle instance named turtle):
>>> turtle.fillcolor(‘violet’)
>>> col = turtle.pencolor()
>>> turtle.fillcolor(col)
>>> turtle.fillcolor(0, .5, 0)

turtle.forward(distance) Aliases: fd Argument: distance—a number (integer or float) Move the turtle forward by the specified distance, in the direction the turtle is headed.

Method Description

Example (for a Turtle instance named turtle):
>>> turtle.position()
(0.00, 0.00)
>>> turtle.forward(25)
>>> turtle.position()
(25.00,0.00)
>>> turtle.forward(-75)
>>> turtle.position()
(-50.00,0.00)

turtle.get_poly()
Return the lastly recorded polygon.
 	
Example (for a Turtle instance named turtle):
>>> p = turtle.get_poly()
>>> turtle.register_shape("myFavouriteShape", p)

turtle.get_shapepoly()
Return the current shape polygon as tuple of coordinate pairs.
 	
Examples (for a Turtle instance named turtle):
>>> turtle.shape("square")
>>> turtle.shapetransform(4, -1, 0, 2)
>>> turtle.get_shapepoly()
((50, -20), (30, 20), (-50, 20), (-30, -20))

turtle.getscreen()
Return the TurtleScreen object, the turtle is drawing on.
So TurtleScreen-methods can be called for that object.
 	
Example (for a Turtle instance named turtle):
>>> ts = turtle.getscreen()
>>> ts
<turtle.TurtleScreen object at 0x0106B770>
>>> ts.bgcolor("pink")

turtle.goto(x, y=None)
Aliases: setpos setposition
 	
Arguments:
x—a number or a pair/vector of numbers
y—a number None
 	
call: goto(x, y) # two coordinates
– or: goto((x, y)) # a pair (tuple) of coordinates
– or: goto(vec) # e.g. as returned by pos()
 	
Move turtle to an absolute position. If the pen is down,
a line will be drawn. The turtle's orientation does not change.
 	
Example (for a Turtle instance named turtle):
>>> tp = turtle.pos()
>>> tp

F

Method Description

(0.00, 0.00)
>>> turtle.setpos(60,30)
>>> turtle.pos()
(60.00,30.00)
>>> turtle.setpos((20,80))
>>> turtle.pos()
(20.00,80.00)
>>> turtle.setpos(tp)
>>> turtle.pos()
(0.00,0.00)

turtle.heading()
Return the turtle's current heading.
 	
Example (for a Turtle instance named turtle):
>>> turtle.left(67)
>>> turtle.heading()
67.0

turtle.hideturtle() Makes the turtle invisible. Aliases: ht It's a good idea to do this while you're in the middle of a complicated drawing, because hiding the turtle speeds up the drawing observably. Example (for a Turtle instance named turtle): >>> turtle.hideturtle()

turtle.isdown()
Return True if pen is down, False if it's up.
 	
Example (for a Turtle instance named turtle):
>>> turtle.penup()
>>> turtle.isdown()
False
>>> turtle.pendown()
>>> turtle.isdown()
True

turtle.isvisible()
Return True if the Turtle is shown, False if it's hidden.
 	
Example (for a Turtle instance named turtle):
>>> turtle.hideturtle()
>>> print(turtle.isvisible())
False

turtle.left(angle) Aliases: lt Argument: angle—a number (integer or float)

Method	Description

	Turn turtle left by angle units. (Units are by default degrees,
	but can be set via the degrees() and radians() functions.)
	Angle orientation depends on mode. (See this.)
	
	Example (for a Turtle instance named turtle):
	>>> turtle.heading()
	22.0
	>>> turtle.left(45)
	>>> turtle.heading()
	67.0

turtle.onclick(fun, btn=1, add=None)	Bind fun to mouse-click event on this turtle on canvas.
	
	Arguments:
	fun—a function with two arguments, to which will be assigned
	the coordinates of the clicked point on the canvas.
	num—number of the mouse-button defaults to 1 (left mouse button).
	add—True or False. If True, new binding will be added, otherwise
	it will replace a former binding.
	
	Example for the anonymous turtle, i.e. the procedural way:
	
	>>> def turn(x, y):
	  turtle.left(360)
	
	>>> onclick(turn) # Now clicking into the turtle will turn it.
	>>> onclick(None) # event-binding will be removed

turtle.ondrag(fun, btn=1, add=None)	Bind fun to mouse-move event on this turtle on canvas. Arguments: fun—a function with two arguments, to which will be assigned the coordinates of the clicked point on the canvas. num—number of the mouse-button defaults to 1 (left mouse button). Every sequence of mouse-move-events on a turtle is preceded by a mouse-click event on that turtle. Example (for a Turtle instance named turtle): >>> turtle.ondrag(turtle.goto) ### Subsequently clicking and dragging a Turtle will ### move it across the screen thereby producing handdrawings ### (if pen is down).
turtle.onrelease(fun, btn=1, add=None)	Bind fun to mouse-button-release event on this turtle on canvas. Arguments: fun—a function with two arguments, to which will be assigned the coordinates of the clicked point on the canvas. num— number of the mouse-button defaults to 1 (left mouse button).

F

Method	Description

turtle.pencolor(*args)	Return or set the pencolor.
	
	Arguments:
	Four input formats are allowed:
	– pencolor()
	Return the current pencolor as color specification string,
	possibly in hex-number format (see example).
	May be used as input to another color/pencolor/fillcolor call.
	– pencolor(colorstring)
	s is a Tk color specification string, such as “red” or “yellow”
	– pencolor((r, g, b))
	a tuple of r, g, and b, which represent, an RGB color,
	and each of r, g, and b are in the range 0..colormode,
	where colormode is either 1.0 or 255
	– pencolor(r, g, b)
	r, g, and b represent an RGB color, and each of r, g, and b
	are in the range 0..colormode
	
	If turtleshape is a polygon, the outline of that polygon is drawn
	with the newly set pencolor.
	
	Example (for a Turtle instance named turtle):
	>>> turtle.pencolor(‘brown’)
	>>> tup = (0.2, 0.8, 0.55)
	>>> turtle.pencolor(tup)
	>>> turtle.pencolor()
	‘#33cc8c’

turtle.pendown()	Pull the pen down—drawing when moving. Aliases: pd down Example (for a Turtle instance named turtle): >>> turtle.pendown()
turtle.pensize(width=None)	Set or return the line thickness. Aliases: width Argument: width—positive number Set the line thickness to width or return it. If resizemode is set to “auto” and turtleshape is a polygon, that polygon is drawn with the same line thickness. If no argument is given, current pensize is returned. Example (for a Turtle instance named turtle): >>> turtle.pensize() 1 turtle.pensize(10) # from here on lines of width 10 are drawn

Method Description
turtle.penup() Pull the pen up—no drawing when moving. Aliases: pu up Example (for a Turtle instance named turtle): >>> turtle.penup()

turtle.radians()
Set the angle measurement units to radians.
 	
Example (for a Turtle instance named turtle):
>>> turtle.heading()
90
>>> turtle.radians()
>>> turtle.heading()
1.5707963267948966

turtle.reset()
Delete the turtle's drawings from the screen, re-center the turtle and set variables to the default values.
 	
Example (for a Turtle instance named turtle):
>>> turtle.position()
(0.00,-22.00)
>>> turtle.heading()
100.0
>>> turtle.reset()
>>> turtle.position()
(0.00,0.00)
>>> turtle.heading()
0.0

turtle.setheading(to_angle)
Set the orientation of the turtle to to_angle.
 	
Aliases: seth
 	
Argument:
to_angle—a number (integer or float)
 	
Set the orientation of the turtle to to_angle.
Here are some common directions in degrees:
 	
standard—mode: logo-mode:
———————————
0 - east 0 - north
90 - north 90 - east
180 - west 180 - south
270 - south 270 - west
 	
Example (for a Turtle instance named turtle):
>>> turtle.setheading(90)
>>> turtle.heading()
90

F

Method Description

turtle.shape(name=None)
Set turtle shape to shape with given name / return current shapename.
 	
Optional argument:
name—a string, which is a valid shapename
 	
Set turtle shape to shape with given name or, if name is not given,
return name of current shape.
Shape with name must exist in the TurtleScreen's shape dictionary.
Initially there are the following polygon shapes:
'arrow', 'turtle', 'circle', 'square', 'triangle', 'classic'.
To learn about how to deal with shapes see Screen-method register_shape.
 	
Example (for a Turtle instance named turtle):
>>> turtle.shape()
'arrow'
>>> turtle.shape("turtle")
>>> turtle.shape()
'turtle'

turtle.showturtle()
Makes the turtle visible.
 	
Aliases: st
 	
Example (for a Turtle instance named turtle):
>>> turtle.hideturtle()
>>> turtle.showturtle()

turtle.speed(speed=None) Return or set the turtle's speed. Optional argument: speed—an integer in the range 0..10 or a speedstring (see below) Set the turtle's speed to an integer value in the range 0 .. 10. If no argument is given: return current speed. If input is a number greater than 10 or smaller than 0.5, speed is set to 0. Speedstrings are mapped to speedvalues in the following way: 'fastest' : 0 'fast' : 10 'normal' : 6 'slow' : 3 'slowest' : 1 speeds from 1 to 10 enforce increasingly faster animation of line drawing and turtle turning. Attention: speed = 0 : *no* animation takes place. forward/back makes turtle jump and likewise left/right make the turtle turn instantly. Example (for a Turtle instance named turtle): >>> turtle.speed(3)

Method Description

turtle.undo()
Undo (repeatedly) the last turtle action.
Number of available undo actions is determined by the size of the undobuffer.
 	
Example (for a Turtle instance named turtle):
>>> for i in range(4):
turtle.fd(50); turtle.lt(80)
 	
>>> for i in range(8):
turtle.undo()

turtle.write(arg, move=False, align='left', font=('Arial', 8, 'normal'))
Write text at the current turtle position.
 	
Arguments:
arg—info, which is to be written to the TurtleScreen
move (optional)—True/False
align (optional)—one of the strings "left", "center" or "right"
font (optional)—a triple (fontname, fontsize, fonttype)
 	
Write text—the string representation of arg—at the current turtle position according to align ("left", "center" or "right") and with the given font.
If move is True, the pen is moved to the bottom-right corner of the text. By default, move is False.
 	
Example (for a Turtle instance named turtle):
>>> turtle.write('Home = ', True, align="center")
>>> turtle.write((0,0), True)

turtle.xcor()
Return the turtle's x coordinate.
 	
Example (for a Turtle instance named turtle):
>>> reset()
>>> turtle.left(60)
>>> turtle.forward(100)
>>> print(turtle.xcor())
50.0

turtle.ycor()
Return the turtle's y coordinate
 	
Example (for a Turtle instance named turtle):
>>> reset()
>>> turtle.left(60)
>>> turtle.forward(100)
>>> print(turtle.ycor())
86.6025403784

TurtleScreen Methods

G

This documentation was generated from the Python documentation available by typing

```
from turtle import *
help(TurtleScreen)
```

in the Python shell. In the documentation found here the variable *turtle* is a reference to a Turtle object and *screen* is a reference to the *TurtleScreen* object. This is a subset of that documentation. To see complete documentation use the Python help system as described above.

Method Description
screen.addshape(name) Same thing as screen.register_shape(name)

screen.bgcolor(*args)
Set or return backgroundcolor of the TurtleScreen.
 	
Arguments (if given): a color string or three numbers
in the range 0..colormode or a 3-tuple of such numbers.
 	
Example (for a TurtleScreen instance named screen):
>>> screen.bgcolor(“orange”)
>>> screen.bgcolor()
‘orange’
>>> screen.bgcolor(0.5,0,0.5)
>>> screen.bgcolor()
‘#800080’

screen.bgpic(picname=None) Set background image or return name of current backgroundimage. Optional argument: picname—a string, name of a gif-file or “nopic”.

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

Method	Description

	If picname is a filename, set the corresponing image as background.
	If picname is “nopic”, delete backgroundimage, if present.
	If picname is None, return the filename of the current backgroundimage.
	
	Example (for a TurtleScreen instance named screen):
	>>> screen.bgpic()
	‘nopic’
	>>> screen.bgpic(“landscape.gif”)
	>>> screen.bgpic()
	‘landscape.gif’

screen.clear()	Delete all drawings and all turtles from the TurtleScreen. Reset empty TurtleScreen to its initial state: white background, no backgroundimage, no eventbindings and tracing on. Example (for a TurtleScreen instance named screen): screen.clear() Note: this method is not available as function.

screen.colormode(cmode=None)	Return the colormode or set it to 1.0 or 255.
	
	Optional argument:
	cmode—one of the values 1.0 or 255
	
	r, g, b values of colortriples have to be in range 0..cmode.
	
	Example (for a TurtleScreen instance named screen):
	>>> screen.colormode()
	1.0
	>>> screen.colormode(255)
	>>> turtle.pencolor(240,160,80)

screen.delay(delay=None)	Return or set the drawing delay in milliseconds.
	
	Optional argument:
	delay—positive integer
	
	Example (for a TurtleScreen instance named screen):
	>>> screen.delay(15)
	>>> screen.delay()
	15

screen.getcanvas()	Return the Canvas of this TurtleScreen.
	
	Example (for a Screen instance named screen):
	>>> cv = screen.getcanvas()
	>>> cv
	<turtle.ScrolledCanvas instance at 0x010742D8>

Method Description
screen.getshapes() Return a list of names of all currently available turtle shapes. Example (for a TurtleScreen instance named screen): >>> screen.getshapes() [‘arrow’, ‘blank’, ‘circle’, . . . , ‘turtle’]
screen.listen(xdummy=None, ydummy=None) Set focus on TurtleScreen (in order to collect key-events) Dummy arguments are provided in order to be able to pass listen to the onclick method. Example (for a TurtleScreen instance named screen): >>> screen.listen()

screen.mode(mode=None)
Set turtle-mode (‘standard’, ‘logo’ or ‘world’) and perform reset.
 	
Optional argument:
mode—on of the strings ‘standard’, ‘logo’ or ‘world’
 	
Mode ‘standard’ is compatible with turtle.py.
Mode ‘logo’ is compatible with most Logo-Turtle-Graphics.
Mode ‘world’ uses userdefined ‘worldcoordinates’. *Attention*: in
this mode angles appear distorted if x/y unit-ratio doesn’t equal 1.
If mode is not given, return the current mode.
 	
Mode Initial turtle heading positive angles
———————————————
‘standard’ to the right (east) counterclockwise
‘logo’ upward (north) clockwise
 	
Examples:
>>> mode(‘logo’) # resets turtle heading to north
>>> mode()
‘logo’

screen.onclick(fun, btn=1, add=None)
Bind fun to mouse-click event on canvas.
 	
Arguments:
fun—a function with two arguments, the coordinates of the
clicked point on the canvas.
num—the number of the mouse-button, defaults to 1
 	
Example (for a TurtleScreen instance named screen
and a Turtle instance named turtle):
 	
>>> screen.onclick(turtle.goto)
 	
### Subsequently clicking into the TurtleScreen will 	
### make the turtle move to the clicked point. 	
>>> screen.onclick(None)
 	
### event-binding will be removed 	

Method Description

screen.onkey(fun, key)

Bind fun to key-release event of key.

Arguments:
fun—a function with no arguments
key—a string: key (e.g. “a”) or key-symbol (e.g. “space”)

In order to be able to register key-events, TurtleScreen
must have focus. (See method listen.)

Example (for a TurtleScreen instance named screen
and a Turtle instance named turtle):

```
>>> def f():
        turtle.fd(50)
        turtle.lt(60)


>>> screen.onkey(f, “Up”)
>>> screen.listen()

### Subsequently the turtle can be moved by
### repeatedly pressing the up-arrow key,
### consequently drawing a hexagon
```

screen.onkeypress(fun, key=None)

Bind fun to key-press event of key if key is given,
or to any key-press-event if no key is given.

Arguments:
fun—a function with no arguments
key—a string: key (e.g. “a”) or key-symbol (e.g. “space”)

In order to be able to register key-events, TurtleScreen
must have focus. (See method listen.)

Example (for a TurtleScreen instance named screen
and a Turtle instance named turtle):

```
>>> def f():
        turtle.fd(50)


>>> screen.onkey(f, “Up”)
>>> screen.listen()

### Subsequently the turtle can be moved by
### repeatedly pressing the up-arrow key,
### or by keeping pressed the up-arrow key.
### consequently drawing a hexagon.
```

Method Description

screen.ontimer(fun, t=0)
Install a timer, which calls fun after t milliseconds.
 	
Arguments:
fun—a function with no arguments.
t—a number >= 0
 	
Example (for a TurtleScreen instance named screen):
 	
>>> running = True
>>> def f():
if running:
turtle.fd(50)
turtle.lt(60)
screen.ontimer(f, 250)
 	
>>> f() ### makes the turtle marching around
>>> running = False

screen.register_shape(name, shape=None) Adds a turtle shape to TurtleScreen's shapelist. Arguments: (1) name is the name of a gif-file and shape is None. Installs the corresponding image shape. !! Image-shapes DO NOT rotate when turning the turtle, !! so they do not display the heading of the turtle! (2) name is an arbitrary string and shape is a tuple of pairs of coordinates. Installs the corresponding polygon shape (3) name is an arbitrary string and shape is a (compound) Shape object. Installs the corresponding compound shape. To use a shape, you have to issue the command shape(shapename). call: register_shape("turtle.gif") –or: register_shape("tri", ((0,0), (10,10), (-10,10))) Example (for a TurtleScreen instance named screen): >>> screen.register_shape("triangle", ((5,-3),(0,5),(-5,-3)))
screen.reset() Reset all Turtles on the Screen to their initial state. Example (for a TurtleScreen instance named screen): >>> screen.reset()

Method Description
screen.screensize(canvwidth=None, canvheight=None, bg=None) Resize the canvas the turtles are drawing on. Optional arguments: canvwidth—positive integer, new width of canvas in pixels canvheight— positive integer, new height of canvas in pixels bg—colorstring or color-tupel, new backgroundcolor If no arguments are given, return current (canvaswidth, canvasheight) Do not alter the drawing window. To observe hidden parts of the canvas use the scrollbars. (Can make visible those parts of a drawing, which were outside the canvas before!) Example (for a Turtle instance named turtle): >>> turtle.screensize(2000,1500) ### e.g. to search for an erroneously escaped turtle ;-)

screen.setworldcoordinates(llx, lly, urx, ury)
Set up a user defined coordinate-system.
 	
Arguments:
llx—a number, x-coordinate of lower left corner of canvas
lly—a number, y-coordinate of lower left corner of canvas
urx—a number, x-coordinate of upper right corner of canvas
ury—a number, y-coordinate of upper right corner of canvas
 	
Set up user coodinat-system and switch to mode 'world' if necessary.
This performs a screen.reset. If mode 'world' is already active,
all drawings are redrawn according to the new coordinates.
 	
But ATTENTION: in user-defined coordinatesystems angles may appear
distorted. (see Screen.mode())
 	
Example (for a TurtleScreen instance named screen):
>>> screen.setworldcoordinates(-10,-0.5,50,1.5)
>>> for _ in range(36):
turtle.left(10)
turtle.forward(0.5)

screen.title(titlestr) Set the title of the Turtle Graphics screen. The title appears in the title bar of the window.

Method Description

screen.tracer(n=None, delay=None)
Turns turtle animation on/off and set delay for update drawings.
 	
Optional arguments:
n—nonnegative integer
delay—nonnegative integer
 	
If n is given, only each n-th regular screen update is really performed.
(Can be used to accelerate the drawing of complex graphics.)
Second arguments sets delay value (see RawTurtle.delay())
 	
Example (for a TurtleScreen instance named screen):
>>> screen.tracer(8, 25)
>>> dist = 2
>>> for i in range(200):
turtle.fd(dist)
turtle.rt(90)
dist += 2

screen.turtles() Return the list of turtles on the screen. Example (for a TurtleScreen instance named screen): >>> screen.turtles() [<turtle.Turtle object at 0x00E11FB0>]
screen.update() Perform a TurtleScreen update.
screen.window_height() Return the height of the turtle window. Example (for a TurtleScreen instance named screen): >>> screen.window_height() 480
screen.window_width() Return the width of the turtle window. Example (for a TurtleScreen instance named screen): >>> screen.window_width() 640
screen.mainloop() Starts event loop—calling Tkinter's mainloop function. Must be last statement in a turtle graphics program. Must NOT be used if a script is run from within IDLE in -n mode (No subprocess)—for interactive use of turtle graphics. Example (for a TurtleScreen instance named screen): >>> screen.mainloop()

Method Description
screen.numinput(title, prompt, default=None, minval=None, maxval=None) Pop up a dialog window for input of a number. Arguments: title is the title of the dialog window, prompt is a text mostly describing what numerical information to input. default: default value minval: minimum value for imput maxval: maximum value for input The number input must be in the range minval .. maxval if these are given. If not, a hint is issued and the dialog remains open for correction. Return the number input. If the dialog is canceled, return None. Example (for a TurtleScreen instance named screen): >>> screen.numinput(“Poker”, “Your stakes:”, 1000, minval=10, maxval=10000)
screen.textinput(title, prompt) Pop up a dialog window for input of a string. Arguments: title is the title of the dialog window, prompt is a text mostly describing what information to input. Return the string input If the dialog is canceled, return None. Example (for a TurtleScreen instance named screen): >>> screen.textinput(“NIM”, “Name of first player:”)

The Reminder! Program

H

```
import sys
import tkinter
import tkinter.messagebox
import os

def addReminder(text,x,y,notes,reminders):
    notewin = tkinter.Toplevel()
    notewin.resizable(width=False,height=False)
    notewin.geometry("+"+str(x)+"+"+str(y))

    reminder = tkinter.Text(notewin,bg=``yellow'', width=30,height=15)

    reminder.insert(tkinter.END,text)

    reminder.pack()

    notes.append(notewin)
    reminders.append(reminder)


    def deleteWindowHandler():
        print("Window Deleted")
        notewin.withdraw()
        notes.remove(notewin)
            reminders.remove(reminder)

    notewin.protocol("WM_DELETE_WINDOW", deleteWindowHandler)


def main():

    def post():
            print("Post")
        addReminder(note.get("1.0",tkinter.END), \
                    root.winfo_rootx()+5,root.winfo_rooty()+5,notes,reminders)
        note.delete("1.0",tkinter.END)

    root = tkinter.Tk()

    root.title("Reminder!")
    root.resizable(width=False,height=False)

    notes = []
    reminders = []

    bar = tkinter.Menu(root)

```

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

```
48      fileMenu = tkinter.Menu(bar,tearoff=0)
49      fileMenu.add_command(label="Exit",command=root.quit)
50      bar.add_cascade(label="File",menu=fileMenu)
51      root.config(menu=bar)
52
53      mainFrame = tkinter.Frame(root,borderwidth=1,padx=5,pady=5)
54      mainFrame.pack()
55
56      note = tkinter.Text(mainFrame,bg=``yellow'', width=30,height=15)
57      note.pack()
58
59      tkinter.Button(mainFrame,text="New Reminder!", command=post).pack()
60
61      try:
62          print("reading reminders.txt file")
63          file = open("reminders.txt","r")
64          x = int(file.readline())
65          y = int(file.readline())
66          root.geometry("+"+str(x)+"+"+str(y))
67
68          line = file.readline()
69          while line.strip() != "":
70              x = int(line)
71              y = int(file.readline())
72              text = ""
73              line = file.readline()
74              while line.strip() != "____....____._._._":
75                  text = text + line
76                  line = file.readline()
77
78              text = text.strip()
79
80              addReminder(text,x,y,notes,reminders)
81
82              line = file.readline()
83      except:
84          print("reminders.txt not found")
85
86
87
88      def appClosing():
89          print("Application Closing")
90          file = open("reminders.txt","w")
91
92          file.write(str(root.winfo_x())+"\n")
93          file.write(str(root.winfo_y())+"\n")
94
95          for i in range(len(notes)):
96              print(notes[i].winfo_rootx())
97              print(notes[i].winfo_rooty())
98              print(reminders[i].get("1.0",tkinter.END))
99
100             file.write(str(notes[i].winfo_rootx())+"\n")
101             file.write(str(notes[i].winfo_rooty())+"\n")
102             file.write(reminders[i].get("1.0",tkinter.END)+"\n")
103             file.write("____....____._._._\n")
104
105         file.close()
106         root.destroy()
107         root.quit()
108         sys.exit()
109
110
111     root.protocol("WM_DELETE_WINDOW", appClosing)
112
113
114     tkinter.mainloop()
115
116 if __name__ == "__main__":
117     main()
```

The Bouncing Ball Program

```
from turtle import *
import tkinter
import random

screenMaxX = 300
screenMaxY = 300
screenMinX = -300
screenMinY = -300

# This is a example of a class that uses inheritance.
# The Ball class inherits from the RawTurtle class.
# This is indicated to Python by writing
# class Ball(RawTurtle):
# That says, class Ball inherits from RawTurtle, which
# means that a Ball is also a RawTurtle, but it is a
# little more than just a RawTurtle. The Ball class also
# maintains a dx and dy value that is the amount
# to move as it is animated.
class Ball(RawTurtle):
      # The __init__ is the CONSTRUCTOR. Its purpose is to
      # initialize the object by storing data in the object. Anytime
      # self.variable = value is written a value is being stored in
      # the object referred to by self. self always points to the
      # current object.
      def __init__(self,cv,dx,dy):
            # Because the Ball class inherits from the RawTurtle class
            # the Ball class constructor must call the RawTurtle class
            # constructor to initialize the RawTurtle part of the object.
            # The RawTurtle class is called the BASE class. The Ball class
            # is called the DERIVED class. The call to initialize the
            # base class part of the object is always the first thing
            # you do in the derived class's constructor.
            RawTurtle.__init__(self,cv)

            # Then the rest of the object can be initialized.
            self.penup()
            self.shape("soccerball.gif")
            self.dx = dx
            self.dy = dy

      # The move method is a mutator method. It changes the data
      # of the object by adding something to the Ball's x and y
      # position.
      def move(self):
```

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science, 227
DOI 10.1007/978-1-84996-537-8, © Springer-Verlag London Limited 2011

```
45              newx = self.xcor() + self.dx
46              newy = self.ycor() + self.dy
47
48              # The if statements below make the ball
49              # bounce off the walls.
50              if newx < screenMinX:
51                    newx = 2 * screenMinX - newx
52                    self.dx = -self.dx
53              if newy < screenMinY:
54                    newy = 2 * screenMinY - newy
55                    self.dy = - self.dy
56              if newx > screenMaxX:
57                    newx = 2 * screenMaxX - newx
58                    self.dx = - self.dx
59              if newy > screenMaxY:
60                    newy = 2 * screenMaxY - newy
61                    self.dy = -self.dy
62
63              # Then we call a method on the RawTurtle
64              # to move to the new x and y position.
65              self.goto(newx,newy)
66
67  # Once the classes and functions have been defined we'll put our
68  # main function at the bottom of the file. Main isn't necessarily
69  # written last. It's simply put at the bottom of the file. Main
70  # is not a method. It is a plain function because it is not
71  # defined inside any class.
72  def main():
73
74        # Start by creating a RawTurtle object for the window.
75        root = tkinter.Tk()
76        root.title("Bouncing Balls!")
77        cv = ScrolledCanvas(root,600,600,600,600)
78        cv.pack(side = tkinter.LEFT)
79        t = RawTurtle(cv)
80        fram = tkinter.Frame(root)
81        fram.pack(side = tkinter.RIGHT,fill=tkinter.BOTH)
82
83        screen = t.getscreen()
84        screen.setworldcoordinates(screenMinX,screenMinY,screenMaxX,
85                                   screenMaxY)
86        t.ht()
87        screen.tracer(20)
88        screen.register_shape("soccerball.gif")
89
90        # The ballList is a list of all the ball objects. This
91        # list is needed so the balls can be animated by the
92        # program.
93        ballList = []
94
95        # Here is the animation handler. It is called at
96        # every timer event.
97        def animate():
98              # Tell all the balls to move
99              for ball in ballList:
100                   ball.move()
101
102             # Set the timer to go off again
103             screen.ontimer(animate)
104
105       # This code creates 10 balls heading
106       # in random directions
107       for k in range(10):
108             dx = random.random() * 3 + 1
109             dy = random.random() * 3 + 1
110             # Here is how a ball object is created. We
```

```
111             # write ball = Ball(5,4)
112             # to create an instance of the Ball class
113             # and point the ball reference at that object.
114             # That way we can refer to the object by writing
115             # ball.
116             ball = Ball(cv,dx,dy)
117             # Each new ball is added to the Ball list so
118             # it can be accessed by the animation handler.
119             ballList.append(ball)
120
121       # This is the code for the quit Button handling. This
122       # function will be passed to the quitButton so it can
123       # be called by the quitButton when it wasPressed.
124       def quitHandler():
125             # close the window and quit
126             print("Good Bye")
127             root.destroy()
128             root.quit()
129
130       # Here is where the quitButton is created. To create
131       # an object we write
132       # objectReference = Class(<Parameters to Constructor>)
133       quitButton = tkinter.Button(fram, text = "Quit", command=quitHandler)
134       quitButton.pack()
135
136       # This is another example of a method call. We've been doing
137       # this all semester. It is an ontimer method call to the
138       # TurtleScreen object referred to by screen.
139       screen.ontimer(animate)
140
141       tkinter.mainloop()
142
143 if __name__ == "__main__":
144       main()
```

Glossary

API An abbreviation for Application Programming Interface An API is a collection of functions that provide some service or services to an application

ASCII Abbreviation for the American Standard Code for Information Interchange

American Standard Code for Information Interchange A widely accepted standard for the representation of characters within a computer

CPU The abbreviation of Central Processing Unit

GUI An abbreviation for Graphical User Interface

I/O device An Input/Output device. The device is capable of both storing and retrieving information

IDE An abbreviation for Integrated Development Environment

Linux A freely available open source operating system originated by Linus Torvalds

Mac OS X An operating system developed and supported by Apple, Inc.

Microsoft Windows An operating system developed and supported by Microsoft Corporation

None A special value which is the only value of its type, the NoneType

Python An interpreted programming language

Tk A windowing toolkit or API available for a variety of operating systems

Wing IDE 101 A freely available IDE for educational purposes available from www.wingware.com

XML A meta-language for describing hierarchically organized data. XML stands for eXtensible Markup Language

XML element One node in an XML file that is delimited by start and end tags

accessor method A method that accesses the data of an object (and returns some of it) but does not change the object

accumulator A variable that is used to count something in a program

accumulator pattern An idiom for counting in a program

activation record An area of memory that holds a copy of each variable that is defined in the local scope of an actively executing function

address The name of a byte within memory Addresses are sequentially assigned starting at 0 and continuing to the limit of the CPU's addressable space

arguments Values passed to a method that affect the action that the method performs

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

assignment statement A fundamental operation of storing a value in a named location in a program

binary A counting system composed of 0's and 1's, the only numbers a computer can store

bit A memory location that can hold a 0 or 1

bool The name of the type for True and False representation in Python

bottom-up design A design process where smaller tasks are implemented first and then the solutions, usually in the form of functions or classes, to these smaller tasks are integrated into a solution for a bigger problem

byte Eight bits grouped together. A byte is the smallest unit of addressable memory in a computer

central processing unit The brain of a computer. Often abbreviated CPU

class A collection of methods that all work with the data of a particular type of object. A class and a type are synonymous in Python

computer An electronic device that can be programmed to complete a variety of data processing tasks

constructor A part of a class that is responsible for initializing the data of an object

debugger A program that lets a programmer set breakpoints, look at variables, and trace the execution of a programmer the programmer is developing

delimiter A special character or characters, usually occurring in pairs that sets some text off from surrounding text

dict A type of value that stores key/value pairs in Python

dictionary A mapping from keys to values. The key can be any hashable object. The value can be any object. Keys within the dictionary must be unique. Values do not have to be unique

event An abstraction used to describe the availability of some input to a program that became available while the program was executing. Event-driven programs are written so they can respond to events when the occur

exception A mechanism for handling abnormal conditions during the execution of a program

file A grouping of related data that can be read by a computer program. It is usually stored on a hard drive, but may be stored on a network or any other I/O device

float The name of the type for real number representation in Python

formal parameter A name given to an argument when it is passed to a function

function A sequence of code that is given a name and may be called when appropriate in a program. A Function is passed arguments so it can perform an appropriate action for the current state of the program

garbage collector A part of the Python interpreter that periodically looks for objects in memory that no longer have any references pointing to them. When such an object is found the garbage collector returns the storage for the object to the available memory for creating new objects

gigabyte $1024 = 2^{10}$ megabytes. Abbreviated GB

guess and check A pattern or idiom that can be used to discover a property of the data a program is working with

hard drive An Input/Output device containing non-volatile storage. The contents of the hard drive are not erased when the power is turned off

hashable A technical term that means that the object can be quickly converted to an integer through some encoding of the data within the object

hexadecimal A counting system where each digit has sixteen different values including 0–9 and A–F

hook A means by which a program allows another program to modify its behavior. The Python interpreter has several hooks that allow a programmer

idiom When used in the context of computer programming, an idiom is a short sequence of code that can be used in certain recurring situations

if-then statement A statement where the evaluation of a condition determines which code will executed

immutable A object that cannot be changed once it is created is said to be immutable. Strings, ints, floats, and bools are examples of immutable types. Lists are not immutable

index An integer used to select an item from a sequence. Indices start at 0 for the first item in a sequence

inheritance The reuse of code in object-oriented programming. The reuse makes sense when there is an is-a relationship between two class. For instance, a Circle is-a Shape

instruction A simple command understood by the CPU. For instance, two numbers can be added together by an instruction

int The name of the integer type in Python

integrated development environment A program that includes an editor and debugger for editing and debugging computer programs

interpreter A program that reads another program and executes the statements found there

iteration Repeating the execution of several statements of a program, more than once. The statement are written once, but a loop construct repeats the execution of the statements when the program executes

kilobyte $1024 = 2^{10}$ bytes. Abbreviated KB

list The name of the type for list representation in Python

loop See iteration

megabyte $1024 = 2^{10}$ kilobytes. Abbreviated MB

memory A random access device that stores a program and data while the program is executing. Frequently memory is called RAM, which stands for Random Access Memory

method A sequence of code that accesses or updates the data of an object. A method is an action we take on an object

module A file containing code in Python. Files or modules may be imported into other modules using an *import* statement. Modules must end in *.py* to be imported

mutator method A method that changes or mutates the data of an object

object A grouping of data and the valid operations on that data

octal A counting system where each digit has eight possibilities including 0–7

operator A method that is not called using the *reference.method*(*arguments*) format

parallel lists A set of two or more lists where corresponding locations within the multiple lists contiain related information. Using parallel lists is a programming technique for maintaining lists of information when there are many values that correspond to one record

polymorphism Literally meaning many forms, polymorphism in computer science refers to the right version of a method being called when the same method occurs in more than one type of object. Python supports polymorphism by dynamically looking up the correct method each time it is called in the object it is called on

predicate A function that returns True or False

python shell An interactive session with the Python interpreter

record A grouping of data in a file (for example several lines in a file) that are related to one entity in some way

recursion When a function calls itself it is said to be recursive. Recursion occurs when the function is executing and either directly or indirectly calls itself

reference A pointer that points to an object. A reference is the address of an object in memory

run-time error An error in a program discovered while the interpreter is executing the program

run-time stack A data structure that is used by Python to execute programs. It is a stack of activation records

scope The area in a program where a variable is defined. Scope becomes a factor when writing functions which define a new local scope. The LEGB rule [3] helps us remember there is local, enclosing, global, and built-in scopes in Python

self A reference that points at the current object when a method is executing. Python makes self point to the object that the method was called on

sequence A grouping of like data that can be iterated over. Lists and strings are sequences

set A container type in Python

short-circuit logic An evaluation strategy where a boolean expression is evaluated from left to right only until the truth or falsity of the expression is determined. Any error condition that may have occurred by evaluating further to the right will not be found if the expression's value is known before the offending part is encountered. For instance $5 > 6$ and $6/0 == 1$ would evaluate to False, and would not raise an exception using short-circuit logic

stack See run-time stack

statement The smallest executable unit in the Python programming language

step into The term used when the debugger stops during the execution of the next instruction at any intermediate computation that is performed

step over The term used when a debugger stops after the next statement is executed. Stepping over does not stop at any intermediate computations

str The name of the type for string representation in Python

subclass A class that inherits from another class called the superclass. A subclass is also called a derived class

superclass A class that was inherited from to make a subclass. A superclass is also called a base class

syntactic sugar The ability to write the same thing in at least two ways in a language, one of which is preferable to the other

syntax error An error in the format of a program. Syntax errors are found by the interpreter before actually running a program

tag A delimiter in an XML file

terabyte $1024 = 2^{10}$ gigabytes. Abbreviated TB

top-down design A design process where details are left until later and the main part of the program is written first calling functions that will eventually take care of the details

tuple An aggregate type in Python

turtle A module in Python that provides an abstraction for drawing pictures

type An interpretation of a group of bytes in memory. Certain operations are valid only for certain types of values

volatile store Refers to the properties of a device. Volatile store loses its contents when the power is turned off

while loop A statement used for indefinite iteration. Indefinite means there is no sequence being iterated over in a while loop. Instead the iteration continues until a condition becomes False

widget An element of a GUI application

word Usually four bytes group together. Typically a word is used to store integers in a computer

References

1. James H. Cross, II, T. Dean Hendrix, and Larry A. Barowski. Using the debugger as an integral part of teaching cs1, 2002.
2. David Flanagan and Yukihiro Matsumoto. *The ruby programming language*. O'Reilly, Sebastopol, CA, 2008.
3. Mark Lutz. *Learning Python*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.
4. Money Magazine. Best jobs in america, 2006. [Online; accessed 1/29/2010; http://money.cnn.com/popups/2006/moneymag/bestjobs/frameset.exclude.html].
5. Alex Martelli. *Python in a Nutshell. A Desktop Quick Reference*; *2nd ed.* Nutshell handbook. O'Reilly, Sebastopol, CA, 2006.
6. Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.
7. The U.S. Consitution Online. Steve mount, 2010. [Online; accessed 1/29/2010; http://www.usconstitution.net/const.html#A2Sec1].
8. Mark Pilgrim. Porting code to python 3 with 2to3, 2010. [Online; accessed 1/29/2010; http://diveintopython3.org/porting-code-to-python-3-with-2to3.html].
9. Arild Stubhaug. *The Mathematician Sophus Lie*. Springer, Berlin, Germany, 2002.
10. Guido van Rossum. Guido's personal home page, 2010. [Online; accessed 1/29/2010; http://www.python.org/~guido/].
11. Brent B. Welch. *Practical programming in Tcl and Tk (3rd ed.)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
12. Wikipedia. Ascii, 2010. [Online; accessed 1/29/2010; http://en.wikipedia.org/wiki/ASCII].
13. Wikipedia. George boole, 2010. [Online; accessed 1/29/2010; http://en.wikipedia.org/wiki/George_Boole].
14. Wikipedia. Logo (programming language), 2010. [Online; accessed 1/29/2010; http://en.wikipedia.org/wiki/Logo_(programming_language)].
15. Wikipedia. W. edwards deming, 2010. [Online; accessed 1/29/2010; http://en.wikipedia.org/wiki/William_Deming].

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science,
DOI 10.1007/978-1-84996-537-8,

Index

K.D. Lee, *Python Programming Fundamentals,* Undergraduate Topics in Computer Science, **239**
DOI 10.1007/978-1-84996-537-8, © Springer-Verlag London Limited 2011