

# THE PYTHON STANDARD LIBRARY

This appendix provides a brief description of all functions and classes from the standard Python library and the custom graphics module that are used in this book.

## Built-in Functions

- **abs(*x*)**  
This function computes the absolute value  $|x|$ .  
**Parameters:** *x* A numerical value  
**Returns:** The absolute value of the argument
- **bytes()**
- **bytes(*x*)**  
This function creates a new bytes sequence. If no argument is given, an empty bytes sequence is created. With an integer argument, it creates a bytes sequence with the given number of zero bytes. If a sequence argument is provided, a bytes sequence is created that contains the elements of that sequence.  
**Parameters:** *x* An integer or a sequence  
**Returns:** The new bytes sequence
- **chr(*x*)**  
This function creates a string containing a single character whose Unicode value is *x*.  
**Parameters:** *x* An integer value  
**Returns:** A string containing the character with Unicode value *x*
- **dict()**
- **dict(*container*)**  
This function creates a new dictionary. If no argument is given, it creates an empty dictionary. If *container* is a dictionary, a duplicate copy of that dictionary is created. Otherwise, if *container* is a sequence of immutable objects, the elements of the container become keys of the new dictionary.  
**Parameters:** *container* A sequence or dictionary from which the new dictionary is created  
**Returns:** The new dictionary
- **float(*x*)**  
This function converts a string or an integer to a floating-point value.  
**Parameters:** *x* A string or numerical value  
**Returns:** A new floating-point value
- **hash(*object*)**  
This function creates a hash value for *object*. Hash values are used to compare dictionary keys.  
**Parameters:** *object* An object of any type  
**Returns:** The integer hash value
- **input()**
- **input(*prompt*)**  
This function obtains a sequence of characters from the user via the keyboard (standard input). If an argument is supplied, the *prompt* string is displayed to the console window (standard output) before characters are input.  
**Parameters:** *prompt* A string displayed as the prompt to the user  
**Returns:** A string containing the characters entered by the user
- **int(*x*)**  
This function converts a number or string to an integer.  
**Parameters:** *x* A string or numerical value  
**Returns:** The new integer object
- **isinstance(*object*, *name*)**
- **isinstance(*object*, *nametuple*)**  
This function determines whether an object is an instance of a class or one of its subclasses. The second argument can be a single class name or a tuple of class names. If a tuple is supplied, the function determines whether the object is an instance of any of the classes specified in the tuple.  
**Parameters:** *object* An object of any type  
*name* A single class name  
*nametuple* A tuple of class names  
**Returns:** True if the object is an instance of any of the classes provided as arguments, False otherwise
- **issubclass(*class*, *name*)**
- **issubclass(*class*, *nametuple*)**  
This function determines whether a class is a subclass of another class. The second argument can

be a single class name or a tuple of class names. If a tuple is supplied, the function determines whether the class is a subclass of any of the classes specified in the tuple.

- **len(*container*)**

This function returns the number of elements in a container.

**Parameters:** *container* A container

**Returns:** An integer indicating the number of elements

- **list()**

- **list(*container*)**

This function creates a new list. With no argument, it creates an empty list. If *container* is a list, a duplicate copy of that list is created; if it is a dictionary, a list containing the keys of the dictionary is created. If the container is a sequence, the new list contains the sequence elements.

**Parameters:** *container* A container whose elements are used to create the new list

**Returns:** The new list

- **max(*arg1, arg2, ...*)**

- **max(*container*)**

This function returns the largest value in a collection. If one argument is supplied and it is a container, the largest element in the container is returned. The container must not be empty. If multiple arguments are supplied, the function returns the largest of those values.

**Parameters:** *container* A container  
*arg1, arg2, ...* Values of any type that are comparable

**Returns:** The largest value in a collection

- **min(*arg1, arg2, ...*)**

- **min(*container*)**

This function returns the smallest value in a collection. If one argument is supplied and it is a container, the smallest element in the container is returned. The container must not be empty. If multiple arguments are supplied, the function returns the smallest of those values.

**Parameters:** *container* A container  
*arg1, arg2, ...* Values of any type that are comparable

**Returns:** The smallest value in a collection

- **next(*iterator*)**

This function returns the next item from a container or iterator.

**Parameters:** *iterator* An iterator object or container that can be used with the for loop

**Returns:** The next item from the iterator or container. If there are no additional items, a `StopIteration` exception is raised

- **open(*filename, mode*)**

This function opens a text or binary file named *filename* and associates it with a file object. A file can be opened for reading, writing, or both reading and writing. When a file is opened for reading, the file must exist or an exception is raised. When a file is opened for writing and the file does not exist, a new file is created; otherwise the contents of the existing file are erased. When a file is opened in append mode and the file does not exist, a new file is created; otherwise new text is appended to the end of the existing file.

**Parameters:** *filename* A string indicating the name of the file on disk

*mode* A string indicating the mode in which the file is opened for a text file: read ("r"), write ("w"), append ("a"), and read/write ("r+"). For a binary file: read ("rb"), write ("wb"), append ("ab"), and read/write ("rb+").

**Returns:** A file object that is associated with the file on disk

- **ord(*char*)**

This function returns the Unicode value for a character.

**Parameters:** *char* A string containing one character

**Returns:** The Unicode value of the character in the string

- **print()**

- **print(*arg1, arg2, ...*)**

- **print(*arg1, arg2, ..., end=string, sep=string, file=fileobj*)**

This function prints its arguments (*arg1, arg2, ...*) to the console window (standard output), separated by the *sep* string and followed by the *end* string. If no argument is supplied, the function simply starts a new line. By default, the arguments are separated by a blank space and the last argument is followed by a newline character (`\n`), which starts a new line. To suppress the new line at the end, specify the empty string as the end string (*end=""*). To use a string

other than a blank space to separate the arguments, specify a new sep string. To print to a text file instead of standard output, specify the file object as the `file=` argument.

**Parameters:** *arg<sub>1</sub>, arg<sub>2</sub>, ...* The values to print  
*end=string* An argument indicating that *string* is to be printed following the last argument value  
*sep=string* An argument indicating that *string* is to be printed between the argument values  
*file=fileobj* An argument indicating the text file object *fileobj* to which the arguments are to be printed

- **range(*stop*)**
- **range(*start*, *stop*)**
- **range(*start*, *stop*, *step*)**

This function creates a sequence container of integer values that can be used with a for statement. The sequence of integers ranges from the *start* value to one less than the *stop* value in increments of the *step* value. If only the *stop* value is supplied, *start* is 0 and *step* is 1. If only the *start* and *stop* values are supplied, *step* is 1.

**Parameters:** *start* An integer indicating the first value in the range  
*stop* An integer indicating a value that is at least one larger than the last value to be included in the range  
*step* An integer indicating the step between each value in the sequence  
**Returns:** An iterator object that can be used with the for statement

- **round(*value*)**
- **round(*value*, *digits*)**

This function rounds a numerical value to a given number of decimal places. If only *value* is supplied, it is rounded to the closest integer.

**Parameters:** *value* The integer or floating-point value to be rounded  
*digits* The number of decimal places  
**Returns:** The argument rounded to the closest integer or to the given number of decimal places

- **set()**
- **set(*container*)**

This function creates a new set. If no argument is supplied, it creates an empty set. If *container* is a set, then a duplicate copy of that set is created; if it is a dictionary, a set containing the keys of the

dictionary is created. Otherwise, if *container* is a sequence, the new set contains the unique elements of the sequence.

**Parameters:** *container* A container whose elements are used to create the new set  
**Returns:** The new set

- **sorted(*container*)**

This function creates a sorted list from the elements in *container*. The elements are sorted in ascending order by default.

**Parameters:** *container* A container whose elements are to be sorted  
**Returns:** The new sorted list

- **str(*object*)**

This function converts an object to a string.

**Parameters:** *object* The object to be converted  
**Returns:** The new string

- **sum(*container*)**

This function computes the sum of the elements of *container*, which must contain numbers.

**Parameters:** *container* A container of numerical values to be summed  
**Returns:** The sum of the container elements

- **super()**

When a method is called on the object that this function returns, the superclass method is invoked.

- **tuple()**
- **tuple(*container*)**

This function creates a new tuple. If no argument is supplied, it creates an empty tuple. If *container* is a tuple, a duplicate copy of that tuple is created; if it is a dictionary, a tuple containing the keys of the dictionary is created. Otherwise, if it is a sequence container, the new tuple contains the sequence elements.

**Parameters:** *container* A container whose elements are used to create the new tuple  
**Returns:** The new tuple

## Built-in Classes

### dict Class

- **d = dict()**
- **d = dict(*c*)**

This function creates a new dictionary. If *c* is a dictionary, a duplicate copy of that dictionary is created. Otherwise, if *c* is a sequence of immutable objects, the sequence elements are the keys of

the new dictionary. If no argument is provided, it creates an empty dictionary.

**Parameters:** *c* A dictionary or sequence from which the new dictionary is created

**Returns:** The new dictionary

- `value = d[key]`

The `[]` operator returns the value associated with *key* in the dictionary. The key must exist or an exception is raised.

- `d[key] = value`

The `[]` operator adds a new *key/value* entry to the dictionary if *key* does not exist in the dictionary. If *key* does exist, *value* becomes associated with it.

- `key in d`

- `key not in d`

The `in/not in` operators determine whether *key* is in the dictionary *d*.

- `len(d)`

This function returns the number of entries in the dictionary.

**Parameters:** *d* A dictionary

**Returns:** An integer indicating the number of dictionary entries

- `d.clear()`

This method removes all entries from the dictionary.

- `d.get(key, default)`

This method returns the value associated with *key*, or *default* if *key* is not present.

**Parameters:** *key* The lookup key  
*default* The value returned when the key is not in the dictionary

**Returns:** The value associated with the key or the default value if the key is not present in the dictionary

- `d.items()`

This method returns a list of the dictionary entries. The list contains one tuple for each entry in the dictionary. The first element of each tuple contains a key and the second element contains the value associated with that key.

**Returns:** A list of tuples containing the dictionary entries

- `d.keys()`

This method returns a list of the dictionary keys.

**Returns:** A list containing the keys in the dictionary

- `d.pop(key)`

This method removes *key* and its associated value from the dictionary.

**Parameters:** *key* The lookup key

**Returns:** The value associated with the key

- `d.values()`

This method returns a list of the dictionary values.

**Returns:** A list containing the values in the dictionary

## list Class

- `l = list()`

- `l = list(sequence)`

This function creates a new list that is empty or contains all of the elements of *sequence*.

**Parameters:** *sequence* A sequence from which the new list is created

**Returns:** The new list

- `value = l[position]`

The `[]` operator returns the element at *position* in the list. The position must be within the legal range or an exception is raised.

- `l[position] = value`

The `[]` operator replaces the element at *position* in the list. The position must be within the legal range or an exception is raised.

- `element in l`

- `element not in l`

The `in/not in` operators determine whether *element* is in the list.

- `len(l)`

This function returns the number of elements in the list.

**Parameters:** *l* A list

**Returns:** An integer indicating the number of elements in the list

- `l.append(element)`

This method appends *element* to the end of the list.

**Parameters:** *element* The element to append

- `l.index(element)`

This method returns the position of *element* in the list. The element must be in the list or an exception is raised.

**Parameters:** *element* The element to locate

**Returns:** The position in the list that contains the element

- **`l.insert(position, element)`**

This method inserts *element* at *position* in the list. All elements at and following the given position are moved down one position.

**Parameters:** *position* Position where the element is to be inserted  
*element* The element to insert

- **`l.pop()`**

- **`l.pop(position)`**

This method removes the last element from the list or the element at *position*. All elements following the given position are moved up one position.

**Parameters:** *position* Position of the element to remove

**Returns:** The removed element

- **`l.remove(element)`**

This method removes *element* from the list and moves all elements following it up one position. The element must be in the list or an exception is raised.

**Parameters:** *element* The element to be removed

- **`l.sort()`**

This method sorts the elements in the list from smallest to largest.

## set Class

- **`s = set()`**

- **`s = set(sequence)`**

This function creates a new set that is empty or a copy of *sequence*. If *sequence* contains duplicate values, only one instance of any value is added to the set.

**Parameters:** *sequence* A sequence from which the new set is created

**Returns:** The new set

- **`element in s`**

- **`element not in s`**

The `in`/`not in` operators determine whether *element* is in the set.

- **`len(s)`**

This function returns the number of elements in the set.

**Parameters:** *s* A set

**Returns:** An integer indicating the number of set elements

- **`s.add(element)`**

This method adds *element* to the set. If the element is already in the set, no action is taken.

**Parameters:** *element* The new element

- **`s.clear()`**

This method removes all elements from the set.

- **`s.difference(t)`**

This method creates a new set that contains the elements in set *s* that are not in set *t*.

**Parameters:** *t* A set

**Returns:** The new set that results from set difference

- **`s.discard(element)`**

This method removes *element* from the set. If the element is not a member of the set, no action is taken.

**Parameters:** *element* The element to be removed from the set

- **`s.intersection(t)`**

This method creates and returns a new set that contains the elements that are in both set *s* and set *t*.

**Parameters:** *t* A set

**Returns:** The new set that results from set intersection

- **`s.issubset(t)`**

This method determines whether set *s* is a subset of set *t*.

**Parameters:** *t* A set

**Returns:** True if *s* is a subset of *t*, False otherwise

- **`s.remove(element)`**

This method removes *element* from the set. If *element* is not in the set, an exception is raised.

**Parameters:** *element* The element to be removed

- **`s.union(t)`**

This method creates and returns a new set that contains all elements in set *s* and set *t*.

**Parameters:** *t* A set

**Returns:** The new set that results from set union

## str Class

- **`s = str()`**

- **`s = str(object)`**

This function creates a new string that is empty or the result of converting *object* to a string.

**Parameters:** *object* The object to be converted

**Returns:** The new string

- **`substring in s`**

- **`substring not in s`**

The `in`/`not in` operators determine whether *substring* is in the string *s*.

- **len(*s*)**  
This function returns the length of the string *s*.  
**Parameters:** *s* A string  
**Returns:** An integer indicating the number of characters in the string
- ***s*.count(*substring*)**  
This method returns the number of non-overlapping occurrences of *substring* in the string *s*.  
**Parameters:** *substring* The string to look for  
**Returns:** The number of occurrences of the substring in the string
- ***s*.endswith(*substring*)**  
This method determines whether string *s* ends with *substring*.  
**Parameters:** *substring* The string to look for  
**Returns:** True if string *s* ends with *substring*, False otherwise
- ***s*.find(*substring*)**
- ***s*.find(*substring*, *begin*)**  
This method returns the lowest index in string *s* where *substring* begins, or -1 if *substring* is not found. If the *begin* argument is supplied, the search is performed on the contents of string *s* within the sequence of characters starting at index *begin* through the end of the string.  
**Parameters:** *substring* The string to look for  
*begin* The starting index within string *s* at which the search begins  
**Returns:** The position where *substring* begins in the string
- ***s*.isalnum()**  
This method tests whether the string *s* consists of only letters and digits and contains at least one character.  
**Returns:** True if both conditions are true, False otherwise
- ***s*.isalpha()**  
This method tests whether the string *s* consists of only letters and contains at least one character.  
**Returns:** True if both conditions are true, False otherwise
- ***s*.isdigit()**  
This method tests whether the string *s* consists of only digits and contains at least one character.  
**Returns:** True if both conditions are true, False otherwise
- ***s*.islower()**  
This method tests whether the string *s* consists of only lowercase letters and contains at least one character.  
**Returns:** True if both conditions are true, False otherwise
- ***s*.isspace()**  
This method tests whether the string *s* consists of only white space characters (blank, newline, tab) and contains at least one character.  
**Returns:** True if both conditions are true, False otherwise
- ***s*.isupper()**  
This method tests whether the string *s* consists of only uppercase letters and contains at least one character.  
**Returns:** True if both conditions are true, False otherwise
- ***s*.lower()**  
This method returns a new string that is the lowercase version of string *s*.  
**Returns:** A new lowercase version of the string
- ***s*.lstrip()**
- ***s*.lstrip(*chars*)**  
This method returns a new version of string *s* in which white space (blanks, tabs, and newlines) is removed from the front (left end) of *s*. If an argument is provided, characters in the string *chars* are removed instead of white space.  
**Parameters:** *chars* A string specifying the characters to be removed  
**Returns:** A new version of the string
- ***s*.replace(*old*, *new*)**  
This method returns a new version of string *s* in which every occurrence of the string *old* is replaced by the string *new*.  
**Parameters:** *old* The substring to be replaced  
*new* The substring that replaces the old substring  
**Returns:** A new version of the string
- ***s*.rstrip()**
- ***s*.rstrip(*chars*)**  
This method returns a new version of string *s* in which white space (blanks, tabs, and newlines) is removed from the end (right end) of *s*. If an



argument is provided, characters in the string *chars* are removed instead of white space.

**Parameters:** *chars* A string specifying the characters to be removed

**Returns:** A new version of the string

- **`s.rsplit(sep, maxsplit)`**

This method returns a list of words from string *s* that are split starting from the right end of the string using substring *sep* as the delimiter. At most *maxsplit* + 1 words are made.

**Parameters:** *sep* A substring specifying the separator used for the split  
*maxsplit* The maximum number of splits

**Returns:** A list of words that results from splitting the string

- **`s.split()`**

- **`s.split(sep)`**

- **`s.split(sep, maxsplit)`**

This method returns a list of words from string *s*. If the substring *sep* is provided, it is used as the delimiter; otherwise, any white space character is used. If *maxsplit* is provided, then only that number of splits will be made, resulting in at most *maxsplit* + 1 words.

**Parameters:** *sep* A substring specifying the separator used for the split  
*maxsplit* The maximum number of splits

**Returns:** A list of words that results from splitting the string

- **`s.splitlines()`**

This method returns a list containing the individual lines of a string split using the newline character `\n` as the delimiter.

**Returns:** A list containing the individual lines split from the string

- **`s.startswith(substring)`**

This method determines whether string *s* begins with *substring*.

**Parameters:** *substring* The substring to look for

**Returns:** True if string *s* starts with *substring*, False otherwise

- **`s.strip()`**

- **`s.strip(chars)`**

This method returns a new version of string *s* in which white space (blanks, tabs, and newlines) is

removed from both ends (front and back) of *s*. If an argument is provided, characters in the string *chars* are removed instead of white space.

**Parameters:** *chars* A string specifying the characters to remove

**Returns:** A new version of the string

- **`s.upper()`**

This method returns a new string that is the uppercase version of string *s*.

**Returns:** A new uppercase version of the string

## File Input/Output

### Common Methods and Functions

The following methods and functions are common to both text and binary files.

- **`open(filename, mode)`**

This function opens a text or binary file named *filename* and associates it with a file object. A file can be opened for reading, writing, or both reading and writing. When a file is opened for reading, the file must exist or an exception is raised. When a file is opened for writing and the file does not exist, a new file is created; otherwise the contents of the existing file are erased. When a file is opened in append mode and the file does not exist, a new file is created; otherwise new text is appended to the end of the existing file.

**Parameters:** *filename* A string indicating the name of the file on disk

*mode* A string indicating the mode in which the file is opened. Modes for a text file are: read ("*r*"), write ("*w*"), append ("*a*"), and read/write ("*r+*"). Modes for a binary file are: read ("*rb*"), write ("*wb*"), append ("*ab*"), and read/write ("*rb+*").

**Returns:** A file object that is associated with the file on disk

- **`f.close()`**

This method closes an open file. It has no effect if the file is already closed.

- **`f.seek(offset, relative)`**

This method moves the file marker to the given byte *offset*. The *relative* argument indicates whether the file marker is to be moved relative to the beginning

of the file (SEEK\_SET), the current position of the file marker (SEEK\_CUR), or the end of the file (SEEK\_END).

**Parameters:** *offset* An integer indicating the number of bytes to move the file marker  
*relative* One of the constants SEEK\_SET, SEEK\_CUR, or SEEK\_END (defined in the os module)  
**Returns:** The new absolute position of the file marker

- **`f.tell()`**  
 This method locates the current position of the file marker.  
**Returns:** The absolute position of the file marker

### Text File Methods

The following methods can be used with text files.

- **`f.read()`**
- **`f.read(num)`**  
 This method reads the next *num* characters from a file opened for reading and returns them as a string. If no argument is supplied, the entire file is read and its contents returned in a single string. An empty string is returned when all characters have been read from the file.  
**Parameters:** *num* An integer indicating the number of characters to be read  
**Returns:** A string containing the characters read from the file
- **`f.readline()`**  
 This method reads the next line of text from a file opened for reading and returns it as a string. An empty string is returned when the end of file is reached.  
**Returns:** A string containing the characters read from the file
- **`f.readlines()`**  
 This method reads the remaining text from a file opened for reading and returns it as a list of strings in which each element of the list contains a single line of text from the file.  
**Returns:** A list of strings containing the lines of text read from the file
- **`f.write(string)`**  
 This method writes *string* to a file opened for writing.  
**Parameters:** *string* The string to be written

### Binary File Methods

The following methods can be used with binary files.

- **`f.read()`**
- **`f.read(num)`**  
 This method reads the next *num* bytes from a binary file opened for reading and returns them as a bytes sequence. If no argument is supplied, the entire file is read and its contents returned in a single bytes sequence.  
**Parameters:** *num* An integer indicating the number of bytes to be read  
**Returns:** A bytes sequence containing the bytes read from the file
- **`f.write(data)`**  
 This method writes a sequence of bytes to a binary file opened for writing.  
**Parameters:** *data* A bytes sequence

## csv Module

### reader Class

- **`r = reader(filename)`**  
 This function creates a new reader object that can be used to iterate over the contents of a CSV file. The file must exist or an exception is raised.  
**Parameters:** *filename* A string containing the name of the CSV file  
**Returns:** The CSV reader object

### writer Class

- **`w = writer(filename)`**  
 This function creates a new writer object that can be used to create a new CSV file. If the file does not exist, a new file is created; otherwise the contents of the existing file are erased.  
**Parameters:** *filename* A string containing the name of the CSV file  
**Returns:** The CSV writer object
- **`w = writerow(row)`**  
 This method adds the next row of column data to the CSV file. The data must be provided as a sequence of strings or numbers.  
**Parameters:** *row* The sequence of strings or numbers comprising the next row



## email.mime.multipart Module

- **m = MIMEMultipart()**  
This function creates and returns a MIME Message object that is used to create e-mail messages.  
**Returns:** A Message object

### Message Class

- **m.add\_header(field, value)**  
This method adds a new header entry to the e-mail message. A header entry consists of a field and its corresponding value.  
**Parameters:** *field* A string containing the name of an e-mail message header field  
*value* A string containing the value associated with the given header field
- **m.attach(content)**  
This method adds content to the body of the e-mail message. *content* must be a MIME content object.  
**Parameters:** *content* A MIME content object that contains the content to be added

## email.mime.text Module

- **MIMEText(data, type)**  
This function creates a new MIME content object that stores the text provided in *data*.  
**Parameters:** *data* A string containing the text to be stored in the MIME content object  
*type* A string containing the MIME encoding type of the text; common values include "plain" and "text"  
**Returns:** A MIME content object

## email.mime.image Module

- **MIMEImage(data)**  
This function creates a new MIME content object that stores the raw data of an image. The new content object is initialized with the supplied *data*.  
**Parameters:** *data* A string containing the raw image data  
**Returns:** A MIME content object

## email.mime.application Module

- **MIMEApplication(data)**  
This function creates a new MIME content object that stores application-specific data such as PDF

documents and spreadsheets. The new content object is initialized with the supplied *data*.

**Parameters:** *data* A string containing the application-specific data  
**Returns:** A MIME content object

## json Module

- **dumps(d)**  
This function converts the entire contents of a dictionary to a JSON-formatted string.  
**Parameters:** *d* A dictionary from which the JSON string is created  
**Returns:** A JSON-formatted string
- **loads(s)**  
This function converts a JSON-formatted string to a dictionary. Numerical values specified in the JSON string are converted to the their appropriate data type: integer or floating-point.  
**Parameters:** *s* A string containing data stored in the JSON format  
**Returns:** A dictionary containing the data obtained from the JSON string

## math Module

- **e**  
This constant is the value of *e*, the base of the natural logarithms.
- **pi**  
This constant is the value of  $\pi$ .
- **acos(x)**  
This function returns the angle with the given cosine,  $\cos^{-1} x \in [0, \pi]$ .  
**Parameters:** *x* A floating-point value between  $-1$  and  $1$   
**Returns:** The arc cosine of the argument, in radians
- **asin(x)**  
This function returns the angle with the given sine,  $\sin^{-1} x \in [-\pi/2, \pi/2]$ .  
**Parameters:** *x* A floating-point value between  $-1$  and  $1$   
**Returns:** The arc sine of the argument, in radians
- **atan(x)**  
This function returns the angle with the given tangent,  $\tan^{-1} x \in (-\pi/2, \pi/2)$ .  
**Parameters:** *x* A floating-point value  
**Returns:** The arc tangent of the argument, in radians

- **atan2(*y*, *x*)**  
This function returns the angle with the given tangent,  $\tan^{-1}(y/x) \in (-\pi, \pi)$ . If *x* can equal zero, or if it is necessary to distinguish “northwest” from “southeast”, use this function instead of `atan(y/x)`.  
**Parameters:** *y*, *x* Two floating-point values  
**Returns:** The angle, in radians, between the points (0, 0) and (*x*, *y*)
- **ceil(*x*)**  
This function returns the smallest integer  $\geq x$ .  
**Parameters:** *x* A floating-point value  
**Returns:** The smallest integer greater than or equal to the argument
- **cos(*x*)**  
This function returns the cosine of an angle given in radians.  
**Parameters:** *x* An angle in radians  
**Returns:** The cosine of the argument
- **degrees(*x*)**  
This function converts radians to degrees.  
**Parameters:** *x* An angle in radians  
**Returns:** The angle in degrees
- **exp(*x*)**  
This function returns the value  $e^x$ , where *e* is the base of the natural logarithms.  
**Parameters:** *x* A floating-point value  
**Returns:**  $e^x$
- **fabs(*x*)**  
This function returns the absolute value  $|x|$  as a floating-point value.  
**Parameters:** *x* A numerical value  
**Returns:** The absolute value of the argument as a floating-point value
- **factorial(*x*)**  
This function returns  $x!$ , the factorial of *x*.  
**Parameters:** *x* An integer  $\geq 0$   
**Returns:** The factorial of the argument
- **floor(*x*)**  
This function returns the largest integer  $\leq x$ .  
**Parameters:** *x* A floating-point value  
**Returns:** The largest integer less than or equal to the argument
- **hypot(*x*, *y*)**  
This function returns the Euclidean norm  $\sqrt{x^2 + y^2}$ .  
**Parameters:** *x*, *y* Two numerical values  
**Returns:** The length of the vector from the origin to the point (*x*, *y*)
- **log(*x*)**
- **log(*x*, *base*)**  
This function returns the natural logarithm of *x* (to base *e*) or, if the second argument is given, the logarithm of *x* to the given base.  
**Parameters:** *x* A number greater than 0.0  
*base* An integer  
**Returns:** The logarithm of the argument
- **log2(*x*)**
- **log10(*x*)**  
This function returns the logarithm of *x* to either base 2 or base 10.  
**Parameters:** *x* A number greater than 0.0  
**Returns:** The logarithm of the argument to either base 2 or base 10
- **radians(*x*)**  
This function converts degrees to radians.  
**Parameters:** *x* An angle in degrees  
**Returns:** The angle in radians
- **sin(*x*)**  
This function returns the sine of an angle given in radians.  
**Parameters:** *x* An angle in radians  
**Returns:** The sine of the argument
- **sqrt(*x*)**  
This function returns the square root of *x*,  $\sqrt{x}$ .  
**Parameters:** *x* A floating-point value  $\geq 0.0$   
**Returns:** The square root of the argument
- **tan(*x*)**  
This function returns the tangent of an angle given in radians.  
**Parameters:** *x* An angle in radians  
**Returns:** The tangent of the argument
- **trunc(*x*)**  
This function truncates a floating-point value *x* to an integer.  
**Parameters:** *x* A floating-point value  
**Returns:** The whole number part of the argument as an integer

## os Module

- **SEEK\_CUR**
  - **SEEK\_END**
  - **SEEK\_SET**
- These constants are used with the seek method to indicate the position from which the file marker

is offset: `SEEK_CUR` indicates the offset is relative to the current position of the file marker, `SEEK_END` is relative to the end of the file, and `SEEK_SET` is relative to the beginning of the file.

- **`chdir(path)`**

This function changes the current working directory to *path*.

**Parameters:** *path* A string containing the absolute or relative name of a directory

- **`getcwd()`**

This function returns the complete path name of the current working directory.

**Returns:** A string containing the name of the current working directory

- **`listdir()`**

- **`listdir(path)`**

This function returns a list containing the names of the entries (files, subdirectories) in the current directory or the directory given by *path*.

**Parameters:** *path* A string containing the absolute or relative name of a directory

**Returns:** A list of strings containing the names of entries in a directory

- **`remove(filename)`**

This function deletes an existing file.

**Parameters:** *filename* A string with the absolute or relative name of an existing file

- **`rename(source, dest)`**

This function renames or moves a file.

**Parameters:** *source* A string with the absolute or relative name of an existing file  
*dest* A string containing the new absolute or relative name for the file

## os.path Module

- **`exists(path)`**

This function determines whether *path* refers to an existing file or directory.

**Parameters:** *path* A string containing the absolute or relative name of a directory or file

**Returns:** True if the file or directory exists, and False otherwise

- **`getsize(path)`**

This function returns the size of a file whose name is specified by *path*.

**Parameters:** *path* A string containing the absolute or relative name of a file

**Returns:** The size of the file in bytes

- **`isdir(path)`**

This function indicates whether *path* refers to a directory.

**Parameters:** *path* A string containing the absolute or relative name of a directory

**Returns:** True if *path* is the name of a directory, False otherwise

- **`isfile(path)`**

This function indicates whether *path* refers to a file.

**Parameters:** *path* A string containing an absolute or relative name of a file

**Returns:** True if *path* is the name of a file, False otherwise

- **`join(path, name)`**

This function appends a file or directory name to a path, including the appropriate path separator for the operating system being used.

**Parameters:** *path* A string containing an absolute or relative path name

*name* A string containing the file or directory name to be appended to the path

**Returns:** A string containing the new path name

## random Module

- **`randint(first, last)`**

This function returns the next pseudorandom, uniformly distributed integer in the range from *first* to *last* (inclusive) drawn from the random number generator's sequence.

**Parameters:** *first, last* The first and last values in the integer range

**Returns:** The next pseudorandom integer  $\geq first$  and  $\leq last$

- **`random()`**

This function returns the next pseudorandom, uniformly distributed floating-point number between 0.0 (inclusive) and 1.0 (exclusive) from the random number generator's sequence.

**Returns:** The next pseudorandom floating-point number  $\geq 0.0$  and  $< 1.0$

## re Module

- `split(pattern, string)`

This function splits the *string* at each occurrence of the given regular expression *pattern* and returns the collection of substrings in a list.

**Parameters:** *pattern* a string containing a regular expression pattern used to determine where the given string is split  
*string* a string to be split based on the given expression pattern

**Returns:** A list containing the substrings or words that result from splitting the string

## shutil Module

- `copy(source, dest)`

This function copies the entire contents of a source file to a directory or a new file.

**Parameters:** *source* A string with the absolute or relative name of an existing file  
*dest* A string containing the absolute or relative name of an existing directory or the new file

## smtplib Module

### SMTP Class

- `c = SMTP(host, port)`

This function creates a new SMTP (Simple Mail Transfer Protocol) object for connecting to an e-mail server in order to send an e-mail message.

**Parameters:** *host* A string containing the host name of the e-mail server  
*port* An integer value that indicates the network port number on which the connection to the e-mail server is made

**Returns:** The SMTP connection object

- `c.starttls()`

This method creates the connection to the e-mail server and turns on secure communications.

- `c.login(username, password)`

This method logs in to the e-mail server using the provided credentials.

**Parameters:** *username* A string containing the user name of the account from which the message will be sent  
*password* A string containing the password of the account

- `c.send_message(msg)`

This method transmits an e-mail message from your computer to the server, which in turn sends it to the appropriate recipients.

**Parameters:** *msg* A correctly constructed Message object that contains header entries and body content for an e-mail message

- `c.quit()`

This method closes the connection to the server and quits the e-mail session.

## sys Module

- `argv`

This variable references the list of string arguments passed to the program via the command line.

- `exit()`

- `exit(message)`

This function terminates the program. If an argument is provided, *message* is displayed before termination.

**Parameters:** *message* A string to be printed to the console window

## time Module

- `sleep(seconds)`

This function pauses the program for a given number of seconds.

**Parameters:** *seconds* The number of seconds to pause the program

- `time()`

This function returns the difference, measured in seconds, between the current time and midnight, Universal Time, January 1, 1970.

**Returns:** The current time in seconds since January 1, 1970

## urllib.parse Module

- `urlencode(params)`

This function creates a string that contains a sequence of URL arguments that are formatted for use in the `urllib.request.urlopen` function. The arguments to be included in the string are specified as key/value pairs in the provided dictionary.

**Parameters:** *params* A dictionary containing the key/value pairs of the URL arguments

**Returns:** A string containing a correctly formatted URL argument sequence

## urllib.request Module

- `r = urlopen(url)`  
This function connects to a web page and opens it for reading. An `HTTPResponse` object is returned, which handles the actual reading.  
**Parameters:** *url* A string containing the universal resource locator (URL) of the web page to be read  
**Returns:** An `HTTPResponse` object

### HTTPResponse Class

- `r.read()`  
This method reads the body of the web page and returns it as a byte sequence.  
**Returns:** A byte sequence containing the entire contents of the web page body
- `r.close()`  
This function closes the `HTTPResponse` object and disconnects from the web page.

## ezgraphics Module

The `ezgraphics` module is based on components from an open source project. Visit the <http://ezgraphics.org> web site for more information about the project and to browse tutorials on using its full range of features.

### GraphicsWindow Class

- `w = GraphicsWindow()`
- `w = GraphicsWindow(width, height)`  
This function creates a new graphics window that contains an empty graphics canvas. The size of the canvas defaults to 400 by 400 pixels unless the width and height values are provided.  
**Parameters:** *width, height* The size in pixels of the canvas contained in the window  
**Returns:** The graphics window object
- `w.canvas()`  
This method returns a reference to the graphics canvas contained in the window.  
**Returns:** A reference to the graphics canvas
- `w.close()`  
This method closes the graphics window and permanently removes it from the desktop. It cannot be used after it has been closed.

- `w.getKey()`  
This method gets and returns the next key pressed by the user. The program pauses and waits for the user to press any key.  
**Returns:** A string indicating which key was pressed
- `w.getMouse()`  
This method gets and returns the location of the next mouse button click on the canvas. The program pauses and waits until the user clicks a mouse button in the canvas area.  
**Returns:** A two-element tuple (*x, y*) containing the canvas coordinates of the mouse pointer when the button was clicked
- `w.hide()`  
This method hides or iconizes the graphics window. The window remains open and usable, but it is not visible on the desktop. If the window is not valid, an exception is raised.
- `w.isValid()`  
This method determines whether the graphics window is valid (opened).  
**Returns:** True if the graphics window is valid (opened) or False if it is closed
- `w.setTitle(title)`  
This method sets the text to be displayed in the title bar of the window.  
**Parameters:** *title* The string to be the window title
- `w.show()`  
This method displays the graphics window on the desktop. If it is not valid, an exception is raised.
- `w.wait()`  
This method keeps the graphics window open and waits for the user to click the “close” button in the title bar or for the program to call the close method.

### GraphicsCanvas Class

- `c.clear()`  
This method clears the canvas by removing all geometric shapes and text that were previously drawn on the canvas.
- `c.drawArc(x, y, diameter, startAngle, extent)`  
This method draws a circular arc on the canvas. The style used to draw the arc is specified using the `setArcStyle` method. The default style (“slice”) draws pie slices.  
**Parameters:** *x, y* The coordinates of the top-left corner of the arc’s bounding square

*diameter* The size of the circle (given as an integer) on which the arc is drawn  
*startAngle* The angle (given in degrees) around the outside of the circle at which the arc begins. An angle of zero degrees is the line that passes through the center of the circle parallel to the *x*-axis  
*extent* The size of the arc given as an angle in degrees

**Returns:** An integer value that uniquely identifies the arc, chord, or pie slice on the canvas

- `c.drawImage(image)`
- `c.drawImage(x, y, image)`

This method draws an image on the canvas. The image is drawn with its top-left corner at position (0, 0) or the given (x, y) position on the canvas. When the (x, y) coordinates are not provided, the canvas is resized to tightly fit around the image.

**Parameters:** *x, y* The coordinates of the top-left corner of the image  
*image* A GraphicsImage object containing the image to be displayed

**Returns:** An integer value that uniquely identifies the image on the canvas

- `c.drawLine(x1, y1, x2, y2)`

This method draws a line between two points.

**Parameters:** *x1, y1* The coordinates of the starting point given as integers  
*x2, y2* The coordinates of the end point given as integers

**Returns:** An integer value that uniquely identifies the line on the canvas

- `c.drawOval(x, y, width, height)`

This method draws an oval on the canvas.

**Parameters:** *x, y* The coordinates of the top-left corner of the bounding rectangle  
*width, height* The width and height of the bounding rectangle

**Returns:** An integer value that uniquely identifies the oval on the canvas

- `c.drawPoint(x, y)`

This method draws a single point on the canvas.

**Parameters:** *x, y* The coordinates of the point given as integers

**Returns:** An integer value that uniquely identifies the point on the canvas

- `c.drawPoly(sequence)`
- `c.drawPoly(x1, y1, x2, y2, x3, y3, ...)`

This method draws a polygon on the canvas using the vertices provided as a series of *x*- and *y*-coordinates. The coordinates can be provided in a sequence container as a single argument, or as multiple arguments.

**Parameters:** *sequence* A list or tuple of integer values that specify the coordinates of the polygon vertices. It must contain at least six values

*x1, y1, x2, y2, x3, y3, ...* The integer coordinates of the polygon's vertices

**Returns:** An integer value that uniquely identifies the polygon on the canvas

- `c.drawRect(x, y, width, height)`

This method draws a rectangle on the canvas.

**Parameters:** *x, y* The coordinates of the top-left corner of the rectangle

*width, height* The width and height of the rectangle

**Returns:** An integer value that uniquely identifies the rectangle on the canvas

- `c.drawText(x, y, text)`

This method draws text on the canvas. The text is drawn relative to the anchor point (x, y). The default position of the anchor point is the northwest corner of the bounding box that surrounds the text. If the string contains newline characters, multiple lines of text will be drawn.

**Parameters:** *x, y* The anchor point coordinates  
*text* The string containing the text to be drawn

**Returns:** An integer value that uniquely identifies the text on the canvas

- `c.drawVector(x1, y1, x2, y2)`

This method draws a line between two points with an arrowhead at the end point.

**Parameters:** *x1, y1* The coordinates of the starting point given as integers  
*x2, y2* The coordinates of the end point given as integers

**Returns:** An integer value that uniquely identifies the vector on the canvas

- `c.height()`

This methods returns the height (vertical size) of the canvas.

**Returns:** The height of the canvas



- **c.width()**  
This method returns the width (horizontal size) of the canvas.  
**Returns:** The width of the canvas
- **c.setAnchor(*position*)**  
This method sets the anchor position used when drawing new text on the canvas. The position is a point on the bounding box that surrounds the text; it is specified as a geographic direction.  
**Parameters:** *position* The anchor position given as a string, which must be one of "n", "s", "e", "w", "nw", "ne", "sw", "se", or "center"
- **c.setArcStyle(*style*)**  
This method sets the style used to draw new arcs on the canvas. An arc can be drawn three ways: as a pie slice in which lines are drawn from the perimeter to the circle's center, as an arc in which only the perimeter section is drawn, or as a chord in which the ends of the arc are connected with a straight line.  
**Parameters:** *style* The arc style given as string; must be one of "slice", "arc", or "chord"
- **c.setBackground(*name*)**
- **c.setBackground(*red*, *green*, *blue*)**  
This method sets the background color of the canvas. The color can be specified by name or by the values of its red, green, and blue components.  
**Parameters:** *name* The color name given as a string  
*red*, *green*, *blue* Integers in the range 0 through 255
- **c.setColor(*name*)**
- **c.setColor(*red*, *green*, *blue*)**  
This method sets both the fill and outline color used to draw new shapes and text on the canvas to the same color. The color can be specified by name or by the values of its red, green, and blue components.  
**Parameters:** *name* The color name given as a string  
*red*, *green*, *blue* Integers in the range 0 through 255
- **c.setFill()**
- **c.setFill(*name*)**
- **c.setFill(*red*, *green*, *blue*)**  
This method sets the color used to fill new shapes drawn on the canvas. The color can be specified by name or by the values of its red, green, and blue components. If no argument is given, the fill color is cleared.  
**Parameters:** *name* The color name given as a string  
*red*, *green*, *blue* Integers in the range 0 through 255
- **c.setFont(*family*, *style*, *size*)**  
This method sets the font used to draw new text on the canvas. A font is specified by three characteristics: *family*, *size*, and *style*.  
**Parameters:** *family* The font name given as a string, which must be one of "arial", "courier", "times", or "helvetica"  
*style* The font style given as a string, which must be one of "normal", "bold", "italic", or "bold italic"  
*size* The point size of the font given as a positive integer
- **c.setHeight(*size*)**  
This method changes the height of the canvas.  
**Parameters:** *size* A positive integer indicating the new height
- **c.setJustify(*style*)**  
This method sets the justification used when drawing multiple lines of text on the canvas.  
**Parameters:** *style* The style given as a string; must be one of "left", "right", or "center"
- **c.setLineWidth(*size*)**  
This method sets the width of new lines drawn on the canvas.  
**Parameters:** *size* An integer value  $\geq 0$
- **c.setLineStyle(*style*)**  
This method sets the style used to draw new lines on the canvas. Lines can be solid or dashed.  
**Parameters:** *style* The line style given as a string, which must be either "solid" or "dashed"
- **c.setOutline()**
- **c.setOutline(*name*)**
- **c.setOutline(*red*, *green*, *blue*)**  
This method sets the color used to draw new lines and text on the canvas. The color can be specified by name or by the values of its red, green, and blue components. If no argument is given, the outline color is cleared.  
**Parameters:** *name* The color name given as a string  
*red*, *green*, *blue* Integers in the range 0 through 255
- **c.setWidth(*size*)**  
This method changes the width of the canvas.  
**Parameters:** *size* A positive integer indicating the new width

**GraphicsImage Class**

- `i = GraphicsImage(filename)`
- `i = GraphicsImage(width, height)`  
This function creates a new graphics image that can be used to create or manipulate an RGB color image. An empty image of a given size can be created or an image can be loaded from a file.  
**Parameters:** *filename* A string containing the name of a GIF or PPM image file  
*width, height* The size of the new empty image  
**Returns:** The graphics image object
- `i.clear()`  
This method clears the image and sets the pixels to be transparent. The size of the image is not changed.
- `i.copy()`  
This method creates a copy of the image.  
**Returns:** The duplicate graphics image object
- `i.getPixel(row, col)`  
This method returns the color of a specified pixel in the image. The color is returned as a three-element tuple containing the red, green, and blue component values.  
**Parameters:** *row, col* The vertical and horizontal coordinates of the pixel  
**Returns:** A tuple containing the color component values
- `i.getRed(row, col)`
- `i.getBlue(row, col)`
- `i.getGreen(row, col)`  
These methods returns the corresponding color component value of a specified pixel in the image.  
**Parameters:** *row, col* The vertical and horizontal coordinates of the pixel  
**Returns:** The color component value as an integer
- `i.height()`  
This method returns the height (vertical size) of the image.  
**Returns:** The height of the image
- `i.save(filename)`  
This method saves a copy of the image to a file in the GIF image format.  
**Parameters:** *filename* A string containing the name of the GIF image file

- `i.setPixel(row, col, red, green, blue)`
- `i.setPixel(row, col, color)`

This method sets the color of a specified pixel in the image. The color can be specified by individual values for its red, green, and blue components or by a three-element tuple that contains the three component values (red, green, blue).

**Parameters:** *row, col* The vertical and horizontal coordinates of the pixel  
*color* A tuple containing three integers in the range 0 through 255  
*red, green, blue* Integers in the range 0 through 255

- `i.width()`  
This method returns the width (horizontal size) of the image.  
**Returns:** The width of the image

**turtle Module**

- `backward(distance)`  
This function moves the turtle distance amount along a straight line in the opposite direction.  
**Parameters:** *distance* The number of pixels to move the turtle
- `clear()`  
This function clears the turtle's drawing from the window, but does not move the turtle.
- `forward(distance)`  
This function moves the turtle *distance* amount along a straight line in the current direction.  
**Parameters:** *distance* The number of pixels to move the turtle
- `goto(x, y)`  
This function moves the turtle to the absolute (x, y) position in the window.  
**Parameters:** *x, y* The coordinates of the window position given as integers
- `heading()`  
This function returns the turtle's current heading in degrees.  
**Returns:** The current heading as a floating-point value
- `hideturtle()`  
This function hides the turtle. The hidden turtle cannot draw on the window.

- **home()**  
This function moves the turtle to the origin (0, 0) and sets the heading to the east.
- **left(*angle*)**  
This function turns the turtle left by *angle* degrees.  
**Parameters:** *angle* The angle given as a floating-point value
- **pencolor(*colorname*)**
- **pencolor(*red*, *green*, *blue*)**  
This function sets the color of the pen the turtle uses to draw on the window. The color can be specified by name or by the values of its red, green, and blue components.  
**Parameters:** *colorname* The color name given as a string  
*red*, *green*, *blue* Integers in the range 0 through 255
- **pendown()**  
This function puts the pen down to draw when the turtle moves.
- **pensize(*width*)**  
This function sets the thickness of the line drawn by the turtle to *width* pixels.  
**Parameters:** *width* The width given as an integer.
- **penup()**  
This function picks up the pen to stop the turtle from drawing when it is moved.
- **reset()**  
This function clears the turtle's drawing from the window, moves the turtle to the origin (0, 0), and sets the heading to the east.
- **right(*angle*)**  
This function turns the turtle right by *angle* degrees.  
**Parameters:** *angle* The angle given as a floating-point value
- **showturtle()**  
This function shows the turtle. The turtle must be visible to draw on the window.