LPTHW, ex 22

Study online at quizlet.com/_3clowk

1.	**-**	Yields the difference of its arguments. The numeric arguments are first converted to a common type.
2.	*****	Yields the product of its arguments. The arguments must either both be numbers, or one argument must be an integer (plain or long) and the other must be a sequence. In the former case, the numbers are converted to a common type and then multiplied together. In the latter case, sequence repetition is performed; a negative repetition factor yields an empty sequence.
3.	**%**	Yields the remainder from the division of the first argument by the second. The numeric arguments are first converted to a common type. A zero right argument raises the ZeroDivisionError exception. The arguments may be floating point numbers, e.g., 3.14%0.7 equals 0.34 (since 3.14 equals 4*0.7 + 0.34.) The modulo operator always yields a result with the same sign as its second operand (or zero); the absolute value of the result is strictly smaller than the absolute value of the second operand [2].
4.	**#**	comment
5.	**+**	Yields the sum of its arguments. The arguments must either both be numbers or both sequences of the same type. In the former case, the numbers are converted to a common type and then added together. In the latter case, the sequences are concatenated.
6.	**abs(x)**	Return the absolute value of a number. The argument may be a plain or long integer or a floating point number. If the argument is a complex number, its magnitude is returned.
7.	**/ and //**	Yield the quotient of their arguments. The numeric arguments are first converted to a common type. Plain or long integer division yields an integer of the same type; the result is that of mathematical division with the 'floor' function applied to the result (rounding down to the nearest integer). Division by zero raises the ZeroDivisionError exception.
8.	**comparisons (<, >, <=, >=, !=, ==, in, is, is not)**	Compare the value of two objects. Need not be the same type. All comparisons have the same priority; yield boolean values (True, False); can be chained arbitrarily (and end once a False event is determined). "is/not" test for object identity - return true only if x and y are the same object
9.	**escape characters**	"\\" followed by any ASCII character. Common ones: \a (ASCII bell); \b (ASCII backspace); \t (tab); \' or \" (single or double quotes); \n (newline)
10.	**file.close()**	Close the file. A closed file cannot be read or written any more. Any operation which requires that the file be open will raise a ValueError after the file has been closed. Calling close() more than once is allowed.
11.	**file.readline([size])**	Read one entire line from the file. A trailing newline character is kept in the string (but may be absent when a file ends with an incomplete line). [6] If the size argument is present and non-negative, it is a maximum byte count (including the trailing newline) and an incomplete line may be returned. When size is not 0, an empty string is returned only when EOF is encountered immediately.
12.	**file.readlines([sizehint])**	Read until EOF using readline() and return a list containing the lines thus read. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read. Objects implementing a file-like interface may choose to ignore sizehint if it cannot be implemented, or cannot be implemented efficiently.
13.	**file.read([size])**	Read at most size bytes from the file (less if the read hits EOF before obtaining size bytes). If the size argument is negative or omitted, read all data until EOF is reached. The bytes are returned as a string object

14. **file.seek(offset[, whence])**	Set the file's current position, like stdio's fseek(). The whence argument is optional and defaults to os.SEEK_SET or 0 (absolute file positioning); other values are os.SEEK_CUR or 1 (seek relative to the current position) and os.SEEK_END or 2 (seek relative to the file's end). There is no return value. For example, f.seek(2, os.SEEK_CUR) advances the position by two and f.seek(-3, os.SEEK_END) sets the position to the third to last.
15. **file.truncate([size])**	Truncate the file's size. If the optional size argument is present, the file is truncated to (at most) that size. The size defaults to the current position. The current file position is not changed.
16. **file.write(str)**	Write a string to the file. There is no return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.
17. **float([x])**	Convert a string or a number to floating point. If the argument is a string, it must contain a possibly signed decimal or floating point number, possibly embedded in whitespace.
18. **From [module] import [names]**	Imports names from a module directly into the importing module's symbol table.
19. **import [module]**	Imports a module
20. **int(x)**	Convert a number or string "x" to an integer, or return "0" if no arguments are given. If "x" is floating point, conversion truncates toward zero. Special rules for string and Unicode objects.
21. **len(s)**	Return the length (the number of items) of an object. The argument may be a sequence (string, tuple or list) or a mapping (dictionary).
22. **\n**	escape character for ASCII Linefeed (newline)
23. **open(name[, mode[, buffering]])**	Open a file, returning an object of the file type described in doc section "File Objects." If the file cannot be opened, IOError is raised.
24. **" " " or ' ' '**	triple-quoted string literal
25. **" " or ' '**	string literal designators
26. **os.path**	This module implements some useful functions on pathnames.
27. **os.path.exists(path)**	Return True if path refers to an existing path. Returns False for broken symbolic links. On some platforms, this function may return False if permission is not granted to execute os.stat() on the requested file, even if the path physically exists.
28. **print**	Evaluates each expression in turn and writes the resulting object to standard output. If an object is not a string, it is converted to a string. A space is written before each object, unless the system believes it is positioned at the beginning of a file (certain rules apply). A '\n' character is written at the end, unless the statement ends with a comma.
29. **pydoc**	documentation module
30. **raw_input([prompt])**	If the prompt argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that.
31. **return**	when placed in the lines calling a function, makes that function output something
32. **round(number[, ndigits])**	Return the floating point value number rounded to ndigits digits after the decimal point. If ndigits is omitted, it defaults to zero. The result is a floating point number. Values are rounded to the closest multiple of 10 to the power minus ndigits; if two multiples are equally close, rounding is done away from 0 (so. for example, round(0.5) is 1.0 and round(-0.5) is -1.0).
33. **% (string and Unicode objects)**	String formatting/interpolation operator. Given "format" % "values", % conversion specifications are replaced with zero or more elements of values. Common conversion types are: d (int), f (float), r (string, converts any python object using repr()), s (string, converts any python object using srt())

34.	**sys**	system module
35.	**sys.argv**	The list of command line arguments passed to a Python script. argv[0] is the script name (it is operating system dependent whether this is a full pathname or not). If the command was executed using the -c command line option to the interpreter, argv[0] is set to the string '-c'. If no script name was passed to the Python interpreter, argv[0] is the empty string.
36.	**target = expression**	Assignment statement. (Re)binds names to values and modifies attributes or items of mutable objects. Evaluates the expression list and assigns the returned value to the target.
37.	**var1, var2, var3 = argv**	unpacks the argv[] list
38.	**"x" + "y"**	Concatenates two sequences of the same type, in this case strings.
39.	**"x" * 10**	When used with a sequence and an integer, sequence repetition is performed.