Original documentation by Hon Wai Lau

> The documentation below is the original of Hon Wai Lau, author of the original project. It is reproduced from Google Code Archive.
>
> **Warning:** The documentation below is included here for reference only and may not be up-to-date.

Documentation of the Formulas question type

Introduction

This question type aims to be generic so that various type of non-trivial questions can be created easily. The variable system and answer boxes allows great flexibility to define question and simplify the job to make complex question. In this documentation, various part of this question type is described, including the formatting, variables, grading, unit system and trial mark sequence.

For creating simple question, see Tutorial.

Question text placeholder

You can insert the subquestion text and answer boxes at a given location using placeholders. To substitute a placeholder by its the contents, the placeholder name needed to be enclosed by `{` and `}` in the text.

Placeholder of the subquestions

This kind of placeholder allows you to insert subquestion in a particular location in the main question.

The placeholders name is a string starting with '#' symbol and followed by any characters of `[A-Z a-z 0-9]` , such as `#1`, `#2a`, `#2b` and `#A`. Subquestion can be labelled by a *placeholder name* in the options called '*Placeholder*'. Note that, by default, the subquestion will be appended at the end if no placeholder is specified. To use them in the main question text, type `{#1}` instead of `#1`, for example:

```
The following is the question 1, part a and part b.
{#1a}
{#1b}
The following is the question 2.
{#2}
```

Placeholder of answer boxes

This kind of placeholder provides a simple way to arrange the answer boxes in the subquestion text.

It is particularly useful when the subquestion requires more than one answer. This flexibility allows you to place the answer boxes in the form of matrix, coordinates, embedding into the question text or any other way you wanted.

The placeholder names are `_0` for the first answer box, `_1` for the second answer box, etc, and `_u` for the unit box. The number of the answer boxes that can be used is the same as the number of elements entered in the Answer. For example, with two answers for one subquestion, the following can be used

```
What is the coordinate of the particles?
(x,y) = ({_0}, {_1}) {_u}
```

The box `{_0}` is actually corresponding to the special variable `_0` in the Grading variables. By default, all missing placeholder are automatically appended at the end. That is `{_0}{_u}` for one answer subquestion, and `{_0}{_1}{_u}` for two answers subquestion.

An exception for the substitution is that there will only be one longer answer box when `{_0}{_u}` are immediately neighbour to each other, and there is only one numerical answer and unit. In this case, the answer and unit box will merge together as one long answer box, and the students are expected to type them in the same box.

Variable system

Variables can be used to substitute question texts, define the answers and specify grading criteria. One of the main purpose of the variables system is to simplify the task to make variation of the questions. Since the variables are generated at the beginning of a quiz, the quiz will be halted if any errors occur. Hence, in

order to minimize the error, the system is designed to only have deterministic variables type, constant length lists and no branching.

Scope of variable

There are four places that you can define and manipulate the variables, each of them have different scope of application.

Random variables	The place to define the variation of variables for the whole questions.
Global variables	The scope includes all (instantiated) random variables.
Local variables	The scope includes all global variables. Note: Each subquestion has its own local variables scope.
Grading variables	The scope includes all local variables, plus special variables (e.g. `_0, _1`) depending on students' response.

Main text	All global variables can be used in the substitution
Subquestion text	All local variables for the subquestion can be used in the substitution
Answer	All local variables for the subquestion can be used in the *expression*
Grading criteria	All grading variables can be used in the *expression*

Variable name

A variable name is a string of alphanumeric characters `[a-z A-Z 0-9 _]` (Numbers and underscore cannot be located at the start of string). Examples of valid variable are

```
x
y1
z_1
foo_bar
```

Variable type

Each variable will be assigned with a type implicitly. A type is either number, string or algebraic variable, plus list of numbers or list of strings. List is defined as the elements enclosed by `[` and `]` . An element in list can be referred using syntax `A[b]` , for example, `[4,5,6][0]` gives the first number, that is, 4. These types are listed below:

Number	A number, for example: 1.2e-3
String	Characters enclosed by two double quotes, for example: "Hello"
List of number	Numbers enclosed by `[` and `]` , for example: `[4,5,6]` , The equivalent short hand syntax are `[4:7]` or `[4:7:1.]`
List of string	Strings enclosed by `[` and `]` , for example: `["A", "B", "C"]`

Algebraic variable	It is simply a set of numbers (see syntax) defined in the non-random variable scope, for example: `{1:100}` .

Random variables

During the quiz creation, each random variables will be assigned by *ONE* value defined by the *expression*. Hence, each student can have their own set of value for the quiz attempt.

A random variable has different syntax than other variables and it only be defined in the field *Random variables*. These variables can be defined by assigning a set of elements or a shuffling of a list. The probability of selecting each element is equal so that each element has equal chance to be drawn. There are three main types of expressions:

Set of elements: A set of elements is elements enclosed by '**{**' and '**}**'. The element can be either a number, a string, a number list, or a string list. For example, the variable below is a set of list of numbers, and the probability for element `[2,3]` is 1/5:

```
F = {
[0,0],
[1,1],
[2,4],
[3,9],
[4,16]
};
```

Set of numbers: A set of numbers is numbers enclosed by '**{**' and '**}**'. You can also specified a range in the format of 'start:stop:interval' , where the numbers satisfied (start + n*interval) < stop, n = 0,1,2,3,..., will be added. If the interval is not specified, value 1 will be used. Note that the last end point is **not** included by definition, but may be include for non-integer due to numerical errors. The set of numbers defined below demonstrate different range syntax. Also, the probability of the elements in each set are 1/8, 1/10 and 1/20 respectively.

```
A = {1,2,3,4,5,6,7,8};
B = {0:1:0.1};
C = {1:10, 10:100:10, 100, 200};
```

Shuffled list: A list can be passed to the shuffle function. For the example below, the instantiated variable S with take one of the permutation of the input list, say `[4,3,2,5]` or `[2,5,4,2]` The probability of each list is therefore 1/4! = 1/24.

```
S = shuffle([2,3,4,5]);
```

Variable assignments

Variable assignments allow you to define and manipulate variables. It can be defined in the option field of global variables, local variables and grading variables.

Expression: It is any combinations of the numbers, variables together with operators and functions listed in the Appendix. Typically, it is just a simple mathematical formula evaluated to a number. The variables used in the expression must be defined before.

Assignment: It is used to assign the evaluated result of an expression to a variable, in the form of '***name*** = ***expression*** ;' .

For loop: It allows a simple iteration in the form of `for(element:list)`. All elements `element` in the `list`will be iterated.

Examples of the assignment:

```
a = 1;            # text after "#" and the end of line will be treated as comment
b = exp(3.333);
d = round(b, 1);  # it can round the number to the desire decimal point
e = 1 + sin(2) + pow(a,2);
A = [1,2,3];
x = A[0];
y = A[a];
z = A[2];
w = A[0] + A[1] + A[2];
m = max(x, y);
distance = sqrt(x*x + y*y + z*z);
theta = atan2(y, x);
smaller = x < y;            # smaller is 1
con = (x < y) + (y < z);    # con is 2
B = fill(3,0);              # B is now [0,0,0]
C = map("sqrt",fill(3,16)); # square root of [16,16,16], so C is now [4,4,4]
s = 0;
for (i:A)  s = s + i;       # s is 6 after the loop
for (i:[0:3]) {
 B[i] = sum(map("+",A,fill(3,i)));  # B will be [6,9,12]
}
p = pick(a+9,"","A","B");   # pick() always choose the first element if index out of
range
u = {-3,-2,-1,1:100};       # u and v define algebraic variables, the numbers are the
points for evaluation
v = {-100:100:1};
```

Note:

- **Important**! **Index out of range cannot be checked by the validation** during question save, so you have to check it yourself. Otherwise, the quiz initialization for some students may fail. Use pick() as a safe variant if applicable.
- Logical true is treated as 1 and false is 0, so the variable `con` above gets the value 2.
- There is no branching, however, you may use the ternary operator `(condition) ? (true) : (false)`for number or the pick() function for general case.
- The available functions are list in the Appendix. There are many additional functions for the numeric list operation, and few for the list of string.

Using variable in text

It is simple to substitute the variables in the text, you only need to enclose the corresponding variables by {and } .

Each text field have a scope of variables. All variables `x` of either number or string in the scope of the text can be used to replace the corresponding placeholder `{x}` in the text.

It is also possible to evaluate an *expression* directly in the text by adding an equal sign at the beginning of the bracket such as `{=x/1000}` . It is easier to use if the named variables are not required. However, no error check will be done unless the question is being instantiated in the quiz. An example is the rescaling of meter to kilometer below:

```
What is the speed of the rocket if it travels with distance {=x/1000} km in {t} s?
```

Answer and grading criteria

For a subquestion to become valid, you must give a mark and define an answer for it. Also, grading critera must be specified in order to check the correctness of a students' response.

Answer type

This question support four answer types. Each type will accept a particular set of numbers, operators, functions and possibly algebraic variables. Depending on the quiz purpose, some or all of these answer types may be used.

Number	You can type in the standard scientific E notation such as: 3.14, 6.626e-34.
Numeric	You can type in numbers and arithmetic operation `+ - * / ^` and `( )` and the constant pi such as: 5+1/2, 2^9, 3pi.
Numerical formula	You can type in everything of numeric plus a set of single variable functions sin(), cos(), tan(), asin(), acos(), atan(), exp(), log10(), ln(), sqrt(), abs(), ceil(), floor() such as: sin(pi/12), 10 ln(2)
Algebraic formula	You can type in every numerical formula and any algebraic variables.

Note that:

- **Students will also need to know these rules in order to input the answer correctly.**
- The possible input have the following relation: Number ⊂ Numeric ⊂ Numerical formula ⊂ Algebraic formula.
- The answer will need a list of string for Algebraic formula and list of number for other answer type.
- "^" in the algebraic formula means "power", not "exclusive or"
- Juxtaposition between numbers or symbols mean multiplication.
- The format check in the quiz interface will show warning sign when the format is wrong for the answer type. It does not give any information about the correct answer.
- All symbols will be treated as algebraic variable in the answer type of algebraic formula. Hence, you may need to hint students what symbols should be used in the question.

Model answer

Depending on the answer type, the answer options will accept either expression that evaluated to a list of number or string (for algebraic formula). The size of the list will determine how many input boxes for this subquestion. If only one answer is required, you can specify a number or string instead of the one element

list.

For the answer type of Number, Numeric and Numerical formula, a list of number (or a single number) is required. Suppose the variables are defined, each line below is a possible answer.

```
pi()
[sin(pi()/2), cos(pi()/2)]
[ans[0], ans[1], ans[2]]
ans
```

For the answer type of Algebraic formula, a list of string (or a single string) is required. Suppose the variables are defined, each line below is a possible answer.

```
"exp(-a x)"
"a x^2 + b y^2"
["a sin(x)", "b cos(x)"]
```

Note that all algebraic variables must be defined in order to be usable in the answer. For the answers above to work, you will need to define the following variables:

```
a = 2;
b = 3;
x = {-100:100:1};
y = {1:100:1};
```

Grading criteria

A grading criteria is required to determine the correctness of the students' response. It requires a expression evaluated to a number in which 0 means false and 1 means true. Typically, it is either the relative error or absolute error with a tolerance level.

For the question with only one answer, the absolute error is simply the different between the model answer and students' response. Hence if the true answer is 3.2 and the students' response is 3.1, then absolute error is `|3.2-3.1| = 0.1` You may want to limit the range of correct response, say, 0.05. In this case, you should select `absolute error < 0.05` Similarly, the relative error is defined by the absolute error divided by the absolute value of model answer. See the `_err` and `_relerr` in grading variables below for more details.

Grading variables

For the typical case, the absolute error and relative error can satisfy most grading criteria. However, sometimes there is a need for other grading criteria.

In the scope of Grading variables, it contains all local variables and **students' response**. With the response from student, you can then define your own grading criteria. The information related to the students' response and model answer is stored in the set of special variables started by underscore as shown below:

Variable name	Description
`_0, _1, _2, ...`	The students' response of each "coordinates". The first "coordinate" is `_0` corresponding to the answer box `{_0}` in the subquestion, etc.

`_a`	It is a list of model answer, which is the answer defined in the answer field.
`_r`	It is a list of students' response with the same size as `_a`. The 0-th element is the same as `_0` , etc.
`_d`	It is a list of the different between each elements, given by `_d = diff(_a,_r);` , see Appendix for the details of the function .
`_err`	Absolute error, using Euclidean norm `|a-r|` , i.e. `_err = sqrt(sum(map("*",_d,_d)));`
`_relerr`	Relative error, divide the absolute error by the norm of model answer `|a-r|/|a|` , i.e.`_relerr = _err/sqrt(sum(map("*",_a,_a)));`

Note:

- The corresponding input boxes of `_0, _1, ...` can be specified as `{_0}, {_1}, ...` in the subquestion text.
- **`_relerr` is NOT defined for algebraic answer!** So the `_err` should be used instead.
- For non-algebraic response, the students' response will be rescaled toward the unit of the model answer. For example, if the model answer is 2m and the students' response is 199cm, then `_a` is `[2]` , while `_r` is `[1.99]` and `_0` is `1.99` . It has no effect if no unit is used.
- For the example of more than one "coordinate": if `_a = [100,100]; _r = [101,102];` , then`_relerr = sqrt(1*1+2*2)/sqrt(100*100+100*100) ≒ 0.0158`
- In this sense, the answer defined in the answer field is only a "model answer" because it may not directly related to the correctness of answer.

Manual grading criteria

In general, other than true or false, the grading criteria can be any number between 0 (all wrong) and 1 (all correct). For the value 1, it means the student can get the full mark for this subquestion (see Grading scheme). A fractional value here can represent the partial correctness of the response. Note that value smaller than 0 is treat as 0 and value bigger than 1 is treated as 1. The following are ony some situations that manual grading criteria is desired:

1. **Multiple correct answer**: Suppose that you have asked a question for a number `x` that is a multiple of 7and `40 < x < 50` . How can you achieve this? As mentioned before, the variable `_0` will store the response when a student input the ansyour, say `42` . The following formula can check whether the response is correct:

```
_0 == 42 || _0 == 49
```

2. **Multiple criteria**: The above example only use a fixed number, so how to grade answer with random variations? Suppose you now want a question with a variable range for each question `a < x < a+10` . To determine the correctness, you need to check two criteria, type the following in the Grading Variables and Grading Criteria box respectively:

```
criterion1 = _0 % 7 == 0;  # whether the remainder is 0. Note that true is 1 and false is 0.
criterion2 = a < _0 && _0 < a+10;   # whether the response is in the desired range.
```

```
criterion1 && criterion2
```

3. **Mark for different accuracy**: Suppose you want to give different marks for the accuracy of the response, say full mark for 1% absolute error and half mark for 1% to 5% absolute error. The following criteria can be used:

```
case1 = _err < 0.01;
case2 = _err < 0.05;
```

```
max (case1, 0.5*case2)
```

Unit system

It allows you to test the knowledge of students about unit. Also, it allows the student to use alternative units for the same answer, provided that they are convertible to each other.

You can specify mark fraction deduction for a wrong unit. The wrong unit here means that the unit that is not convertible to the correct unit, under conversion rules. Suppose a student get 2 marks for this subquestion answer. If the unit penalty = 0.2 and she give wrong unit, then the student can get 2(1-0.2) = 1.6 mark for this submission. (See Grading scheme).

If a teacher does not specify any unit for the subquestion, then the unit box will not be displayed and there is no need for students to enter it. However, if the student enters anything after their answer, it is usually considered to be incorrect so the mark of unit will be deduced. Hence, in the above case, it is better to set 1 for mark deduction.

Nevertheless, if you do not want to penalize the student entering arbitrary string at the end of answer, set 0 for mark deduction. i.e. their unit does not count toward their grade, but you still allow student use any convertible unit. Note that you have assumed a default answer that does not require unit.

Unit and format

A *unit* is either a *base unit* or a *composite unit*:

- *Base unit*: a simple alphabetic string without exponent (e.g. ^2)
- *Composite unit*: compose of *base unit* separated by space, possibly together with exponents. Note that the ordering of base unit is not important

For a given physical quantity, the unit name can be entered using the following rules:

1. Base unit: e.g. length (m), it should be simply entered as `m`
2. Positive exponent: e.g. area (m^2), it should be entered as `m^2`.
3. Negative exponent: e.g. wavenumber (m^{-1}), it should be entered as `m^(-1)`.
4. Compose unit: e.g. velocity (m s^{-2}), it should be entered as `m s^(-2)` with the base unit separated by the space (representing multiplication).
5. Alternative for negative exponent: One preceding division operator can also be used, say `m/s^2` which is equivalent to `m s^(-2)`.

Students need to enter the same format for unit (i.e. they need to know the above rule). Examples:

Unit	Description
m	a base unit, meter, for the dimension length

kg	a base unit, kilogram, for the dimension mass
N	a base unit (it is called derived unit in SI), newton, for the physical quantity force
m/s	a composite unit, for the physical quantity velocity
kg m/s^2	a composite unit, with the same dimension as newton

Examples of answer with unit:

```
1 m
0.1 m^2
20 m s^(-1)
400 kg m/s
100 kW
```

Conversion rules

Conversion rules allow students to use alternative unit. For the details of the rule, See UnitConversion.

Other rules: You can specify the conversion of the *base unit* by equating their values under different units. Examples:

```
1 m = 100 cm = 1000 mm;
1 cm = 0.3937 inch;
```

With the above conversion rule, the following responses are completely equivalent:

```
10 inch
25.4 cm
254 mm
0.254 m
```

For the SI prefix, an alternative syntax can be used: *base unit* followed by a list of SI prefix name.

```
W: k M G T;
```

which is equivalent to `1 W = 1e-3 kW = 1e-6 MW = 1e-9 GW = 1e-12 TW` .

Basic conversion rule: For the conversion between common SI unit with different prefix, there are a set of predefined conversion rules for them. The only thing you need to do is to choose the '*Common SI unit*' in the '*Basic conversion rule*'.

The following answers in the each group

```
5 s
5000 ms
5e9 ns
```

```
0.2 m/s
200 mm/s
```

```
1 m^2
10000 cm^2
1e-6 km^2
```

are all equivalent to each other. So if **one** of the answer is correct, all other are also correct answers.

Note that you may also use 'None' if you find it contradicting with your own rules. If you want to define you own, see the file conversion_rules.php.

Identifying composite unit: Note that, the *base units* and *composite unit* cannot be identified in the conversion rules above. To specify a list of allowable *composite unit*, you can use separator '=' in the '*Unit*'. For example

```
N = kg m s^(-2)
```

It identity the unit Newton to its SI counterpart. Hence, both *composite unit* are correct.

Trial mark sequence

This option only apply to the adaptive mode of moodle quiz.

In the adaptive mode, students are allowed to submit answer to a particular question again and again. This field defines the mark sequence that a student can get for each resubmission. Note that it actually alter the default behavior of the adaptive mode.

The input is a list of numbers separated by comma. Each number represents a fraction of the maximum mark that a student can get in the first, second, third, etc. submission. Hence, if this field has value 1, the student can try it once only.

If the sequence is ended with a comma, infinite resubmission is allow. In this case, the mark that students can get in the following unlisted trial is decreased uniformly with the value equal to the difference of last two value. Note that the minimum mark is always zero (See grading scheme).

Trial mark sequence	**Maximum mark for each trial**	**Description**
1	100%	Only one trial is allowed
1, 0.7, 0.3	100%, 70%, 30%	Three trials are allowed
1, 0.7, 0.3, 0	100%, 70%, 30%, 0%	Four trials are allowed, but the last trial has no mark
1, 0, 0, 0	100%, 0%, 0%, 0%	Four trials are allowed, but only the first trial has non-zero mark
1,	100%, 100%, 100%, ...	Infinite trials. The difference is 0, which is repeated
1, .9,	100%, 90%, 80%, 70%, 60%, ...	Infinite trials. Difference between 1st and 2nd trial is 10%, which is repeated
1, .6,	100%, 60%, 20%, 0%, 0%, ...	Infinite trials. Difference between 1st and 2nd trial is 40%, which is repeated
1, .5, .3,	100%, 50%, 30%, 10%, 0%, 0%, ...	Infinite trials. Difference between 2nd and 3rd trial is 20%, which is repeated

Grading scheme

The following is the grading formula used to grade a particular subquestion:

Symbol	Description
c	Correctness. It takes value between 0 and 1. Boolean false is treated as 0 and true is treated as 1. Other values may be possible if manual condition is used (see Grading criteria)
u	Deduction for wrong unit (see Unit system). In the formula, it always takes value 0 if the unit is correct under Conversion rules
m	Default mark of the subquestion.
r_n	Maximum mark fraction of the n-th submission, for adaptive mode only (See Trial mark sequence)
f	The computed final mark.

For non-adaptive mode:

```
f = m*c*(1-u)
```

For adaptive mode (see moodle documentation):

```
f = max(r_n*m*c*(1-u))
```

The maximum is taken over all submissions. From the above formula, even though a student get a low mark in the first attempt, it is still possible for them to get a higher mark in the following attempt.

Appendix

List of functions for number

`pi()`				
`abs(x)`	`acos(x)`	`acosh(x)`	`asin(x)`	`asinh(x)`
`atan(x)`	`atanh(x)`	`ceil(x)`	`cos(x)`	`cosh(x)`
`deg2rad(x)`	`exp(x)`	`expm1(x)`	`floor(x)`	`is_finite(x)`
`is_infinite(x)`	`is_nan(x)`	`log(x)`	`log10(x)`	`log1p(x)`
`rad2deg(x)`	`round(x)`	`sin(x)`	`sinh(x)`	`sqrt(x)`
`tan(x)`	`tanh(x)`			
`atan2(y,x)`	`fmod(x,y)`	`log(x,base)`	`pow(x,y)`	`round(x,precision)`
`min(x1,x2,...)`	`max(x1,x2,...)`			

Note that the list of function used in the variable assignment is (almost) a superset of the list of function of usable in the algebraic formula.

List of operators for number

+	-	*	/	%
>	<	>=	<=	==
!=	&&	| |	!	<<
>>	^	~	&	|
? :				

Note that the operator "^" means "exclusive or" which have different meaning in the algebraic formula. In algebraic formula, "^" means "power".

List of additional functions for non-number type variables

function	Examples	Description

`concat(X1,X2,...)`	`concat([1],[2,3],[4])`
	`# [1,2,3,4]`
	`concat(["A"],["B"])`
	`# ["A","B"]`
		return a list by concatenating X1, X2, ... together

`diff(X,Y)`
`diff(X,Y,N)`
	`diff([1,1,1],[1,4,9])`
	`# [0,3,8]`
	`diff(["x*x"],["x^2"])`
	`# [0]`
	`diff(["x"],["x^2"],20)`
	# A list with number >= 0
		*** return a list of number D of difference between the list X Y elementwise.
		If X and Y are a list of number, then `D[i]` will be`abs(X[i]-Y[i])`
		If X and Y are a list of string, then `D[i]` will be the differenc between the algebraic formula `X[i]` and `Y[i]`

`fill(N, value)`	`fill(3,0)` `# [0,0,0]`	return a list with size `N` and all elements filled by value
`len(X)`	`len([0:10])` `# 10`	return the length of the list X
`inv(X)`	`inv([0,3,1,2])` `# [0,2,3,1]`	return the inverse permutation of X, which is the same as R in `for(i:[0:N]) R[X[i]]=i` with size N of X. It has the property that `X==inv(inv(X))` and`A==sublist(sublist(A,X),in`

`join(s,A,B...)`	`join(" ", ["Hello","world"])`
	`# "Hello world"`
	`join("",1,"+",2, ["="])`
	`# "1+2="`
		return a string that join all elements together and separated
		string `s` .
		The `A, B, ...` can be any number or string or list.

`map(uop,X)`
`map(bop,X,Y)`
	`map("abs", [-1,2,-3])`
	`# [1,2,3]`
	`map("+",[1,2,3],5)`
	`# [6,7,8]`
	`map("<",[3,4], [1,7])`
	`# [0,1]`
		return a list by applying the operator/function elemenwise t
		elements in X.
		The uop or bop are the string of unary or binary operators (
		functions) for numbers respectively.
		`map(uop,X)` has the meaning as R in `for(i:[0:N]) R[i]=op(X[i]);`
		`map(bop,X,Y)` has the meaning as R in `for(i:[0:N]) R[i]=op(X[i],Y[i])` for function,
		or `for(i:[0:N]) R[i]=X[i] bop Y[i]` for operator.
		Either X or Y can be a number and it will automatically conv
		the same size of the other list.

`pick(i,X)`
`pick(i,x0,x1,...)`
	`pick(1<2,["A","B"])`
	`# "B"`
	`pick(10,"","A","B")`
	`# ""`
	`pick(1,[1,2],[3,4])`
	`# [3,4]`
		A safe variant of list subscript. It will pick the i-th element fr
		the list X or the first element if index out of range.
		`pick(i,x0,x1,...)` will pick the i-th element in the`[x0,x1,...]` , or x0 if index out of range.

`sort(X)`
`sort(X,Y)`
	`sort(["B","C","A"])`
	`# ["A","B","C"]`
	`sort(["B","C","A"], [0,2,1])`
	`# ["B","A","C"]`
		return the sorted list of X in ascending order.
		`sort(X,Y)` will sort the list X,Y together by sorting Y in asc
		order and return the sorted list X.

`str(x)`	`str(1.1)` `# "1.1"`	convert the number x into a string

`sublist(X,Y)`	`sublist([7,8,9], [1,1,2])`
	`# [8,8,9]`
	`sublist([7,8,9], [2,1,0])`
	`# [9,8,7]`
		return a sublist R of X using list Y as index, where `R[i] = X[Y[i]]` .
		If Y is a *permutation*, say `Y=shuffle([0:5])`, then X will be
		permuted depending on Y

`sum(X)`	`sum([1,2,3])` `# 6`	return the sum of all element in the list of number X

*** **This function depends on context variables** in addition to the function parameters. It is also the core function to compare the students' response and model answer (see Grading variables). If input X and Y are a list of string, then `X[i]` and `Y[i]` are treated as algebraic formula and all variables in them must be defined before the location of evaluation. For example,

```
a = 3;
x = {1:100};
d = diff(["a x"],["a x^2"]);
```

Please note that the actual algebraic formulas should be "3 x" and "3 x^2" in the above case. In the above evaluation, the d will take a finite value but not close to zero because the algebraic formula are different. In general, any evaluation failure between two algebraic formula will result in a infinite value INF, so that the expression, say, sum(d) < 0.01 will always be false.

The evaluation will take place at N randomly selected points defined in all algebraic variables. The result will be the root mean square difference at all evaluation points $(1/N)\Sigma_i(X_i-Y_i)^2$, which will converge when N tends to infinity. The N is 100 by default if it is not specified.