2.4 Arithmetic expressions (general)

Basics

An ***expression*** is a combination of items, like variables, literals, operators, and parentheses, that evaluates to a value, like 2 * (x + 1). A common place where expressions are used is on the right side of an assignment statement, as in y = 2 * (x + 1).

A ***literal*** is a specific value in code like 2. An ***operator*** is a symbol that performs a built-in calculation, like +, which performs addition. Common programming operators are shown below.

Table 2.4.1: Arithmetic operators.

Arithmetic operator	Description
+	The ***addition*** operator is **+**, as in x + y.
-	The ***subtraction*** operator is **-**, as in x - y. Also, the - operator is for ***negation***, as in -x + y, or x + -y.
*	The ***multiplication*** operator is ***, as in x * y.
/	The ***division*** operator is **/**, as in x / y.

Feedback?

PARTICIPATION ACTIVITY 2.4.1: Expressions.

Indicate which are valid expressions. x and y are variables, and are the only available variables.

1) x + 1

- (•) Valid
- ( ) Not valid

Correct

The text consists of a variable (x), operator (+), and literal (1). If x is 5, the expression evaluates to 5 + 1 or 6.

2) 2 * (x - y)

Correct

- Valid
- Not valid

The text consists of a literal (2), two operators (*, -), and two variables (x, y), properly combined with parentheses. If x is 7 and y is 3, the expression evaluates to 2 * (7 - 3), or 2 * (4), so 8.

3) x

- (•) Valid
- ( ) Not valid

Correct

An expression can just be a variable.

4) 2

- (•) Valid
- ( ) Not valid

Correct

An expression can just be a literal.

5) 2x

- ( ) Valid
- (•) Not valid

Correct

In programming, multiplication typically must be indicated explicitly using the * operator. Abutment is allowed in math, but not usually in programming.

6) 2 + (xy)

- ( ) Valid
- (•) Not valid

Correct

In programming, doing multiplication via abutment, as in xy, is not usually allowed, because xy could be the name of another variable.

7) x - -2

- (•) Valid
- ( ) Not valid

Correct

The first - represents subtraction, while the second - represents negation. If x were 5, the expression would evaluate to 7.

Feedback?

PARTICIPATION ACTIVITY 2.4.2: Capturing behavior with an expression.

Does the expression correctly capture the intended behavior?

1) 6 plus numItems:

```
6 + numItems
```

- (•) Yes
- ( ) No

Correct

Straightforward addition.

2) 6 times numItems:

```
6 x numItems
```

- Yes
- No

Correct

The multiplication operator is *, not x.

3) totDays divided by 12:

```
totDays / 12
```

- Yes
- No

Correct

Straightforward division.

4) 5 times i:

```
5i
```

- Yes
- No

Correct

Abutment not allowed. Requires 5 * i.

5) The negative of userVal:

```
-userVal
```

- Yes
- No

Correct

- serves as subtraction, but also negation.

6) n factorial

```
n!
```

- Yes
- No

Correct

Most languages don't use the ! symbol for factorial.

Feedback?

Evaluation of expressions

An expression ***evaluates*** to a value, which replaces the expression. Ex: If x is 5, then x + 1 evaluates to 6, and y = x + 1 assigns y with 6.

An expression is evaluated using the order of standard mathematics, such order known in programming as ***precedence rules***, listed below.

Table 2.4.2: Precedence rules for arithmetic operators.

Operator/Convention	Description	Explanation
( )	Items within parentheses are evaluated first	In 2 * (x + 1), the x + 1 is evaluated first, with the result then multiplied by 2.
unary -	- used for negation (unary minus) is next	In 2 * -x, the -x is computed first, with the result then multiplied by 2.
*** / %**	Next to be evaluated are *, /, and %, having equal precedence.	(% is discussed elsewhere)
+ -	Finally come + and - with equal precedence.	In y = 3 + 2 * x, the 2 * x is evaluated first, with the result then added to 3, because * has higher precedence than +. Spacing doesn't matter: y = 3+2 * x would still evaluate 2 * x first.
left-to-right	If more than one operator of equal precedence could be evaluated, evaluation occurs left to right.	In y = x * 2 / 3, the x * 2 is first evaluated, with the result then divided by 3.

Feedback?

PARTICIPATION ACTIVITY 2.4.3: Evaluating expressions.

1 2 3 4 2x speed

x = 4

w = 2

y = 3 * (x + 10 / w)

10 / 2

5

3 * (x + 5)

4 + 5

9

Preferred

y = 3 * (x + (10 / w))

y = 3 * 9
27

Many programmers prefer to use parentheses to make order of evaluation more clear when such order is not obvious.

Feedback?

PARTICIPATION ACTIVITY 2.4.4: Evaluating expressions and precedence rules.

Select the expression whose parentheses match the evaluation order of the original expression.

1) y + 2 * z
   - (y + 2) * z
   - y + (2 * z)

Correct

* has precedence over +, so is evaluated first.

2) z / 2-x
   - (z / 2) - x
   - z / (2 - x)

Correct

/ has precedence over -.

3) x * y * z
   - (x * y) * z
   - x * (y * z)

Correct

The two operators have equal precedence, so evaluation occurs left-to-right.

4) x + 1 * y/2
   - ((x + 1) * y) / 2
   - x + ((1 * y) / 2)
   - x+ (1 * (y / 2))

Correct

* and / have precedence over + so will be evaluated first. * and / have equal precedence so are evaluated left-to-right, despite what the original spacing implied.

5) x / 2 + y / 2
   - ((x / 2) + y) / 2
   - (x / 2) + (y / 2)

Correct

/ has precedence over +. The two / are evaluated left-to-right.

6) What is totCount after executing the following?

```
numItems = 5;
totCount = 1 + (2 *
numItems)  * 4;
```

   - 44
   -

Correct

After (2 * 5) is evaluated, * 4 is evaluated because * has precedence over +.

41

Feedback?

Using parentheses to make the order of evaluation explicit

*A common error is to omit parentheses and assume a different order of evaluation than actually occurs, leading to a bug. Ex: If x is 3, then 5 * x+1 might appear to evaluate as 5 * (3+1) or 20, but actually evaluates as (5 * 3) + 1 or 16 (spacing doesn't matter). Good practice is to use parentheses to make order of evaluation explicit, rather than relying on precedence rules, as in: y = (m * x) + b, unless order doesn't matter as in x + y + z.*

Example: Calorie expenditure

A website lists the calories expended by men and women during exercise as follows (source):

Men: Calories = [(Age × 0.2017) — (Weight × 0.09036) + (Heart Rate × 0.6309) — 55.0969] × Time / 4.184

Women: Calories = [(Age × 0.074) — (Weight × 0.05741) + (Heart Rate × 0.4472) — 20.4022] × Time / 4.184

Below are those expressions written using programming notation:

```
caloriesMan = ( (ageYears * 0.2017) - (weightPounds * 0.09036) + (heartBPM * 0.6309) - 55.0969 ) * timeSeconds / 4.184

caloriesWoman = ( (ageYears * 0.074) - (weightPounds * 0.05741) + (heartBPM * 0.4472) - 20.4022 )* timeSeconds / 4.184
```

PARTICIPATION ACTIVITY 2.4.5: Converting a formatted expression to a program expression.

Consider the example above. Match the changes that were made.

[ ] × **Spaces in variable names** –

	Replaced by ( )
	Single words
	-
	*

Reset

Feedback?