Statements

ArrayBase

Begin Case/Case/End Case

Format

```
begin case
   case boolean_expression1
      statements
   case boolean_expression2
      statements
   ...
   else
      statements
end case
```

Description

The `begin case` statement allows a program to test multiple boolean expressions, each specified in a `case` block. When a boolean expression to evaluate to true, the statements in the corresponding `case` block are executed and the remaining `case` blocks are skipped. If no boolean expression evaluates to true then the statements in the `else` block are executed. There can be one or more `case` blocks. The `else` block is optional.

Example

The following program compares two numbers using the `begin case` statement.

```
input "Input a number: ", a
input "Input another number: ", b
begin case
   case a < b
      print a + " is less than " + b
   case a > b
      print a + " is greater than " + b
   else
      print a + " is equal to " + b
end case
```

The above program will display the following output.

```
Input a number: 5
Input another number: 6
5 is less than 6

Input a number: 6
Input another number: 5
6 is greater than 5

Input a number: 5
Input another number: 5
5 is equal to 5
```

Dim

Do/Until Loop

Format

```
do
   statements
until boolean_expression
```

Description

The statements within a do/until loop are executed one or more times until the boolean_expression evaluates to true. The boolean_expression is tested each time after all the statements within the do/until loop are executed.

Example

The following program uses the do/until loop to print a message as many times as the user specifies.

```
input "How many Hellos? ", howmany
index = 1
do
   print "Hello " + index
   index = index + 1
until (index > howmany)
print "Bye!"
```

The above program will work as follows.

```
How many Hellos? 5
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Bye!
```

Learn More: Continue do

Format

```
do
    statements
    continue do
    statements
until boolean_expression
```

Description

The `continue do` statement forces the program to skip over rest of the statements within a `do/until` loop and test the `boolean_expression`.

Example

The following program greets all names except "Mickey" until the name "Foo" is entered.

```
do
    input "Who? ", name$
    if (name$ = "Mickey") then
      continue do
    end if
    print "Hello " + name$
until (name$ = "Foo")
print "Bye!"
```

Learn More: Exit do

Format

```
do
    statements
    exit do
    statements
until boolean_expression
```

Description

The `exit do` statement forces the program to exit the `do/until` loop.

Example

The following program greets all names until the name "Foo" is entered. If the name "Mickey" is entered, then the program exits the `do/until` loop without greeting "Mickey".

```
do
   input "Who? ", name$
   if (name$ = "Mickey") then
     exit do
   end if
   print "Hello " + name$
until (name$ = "Foo")
print "Bye!"
```

If/then/else

Fill

For Loop

Default format

```
for variable = start_expression to stop_expression
  statements
next variable
```

Description

The `for` loop repeats a block of statements a certain number of times. At the beginning of the `for` loop, an index `variable` is initialized to `start_expression`. At the end of each iteration (when the `next` statement is encountered), the `variable` increments by one (by default). The `for` loop stops once the `variable` becomes greater than the `stop_expression`.

Example

```
for i = 1 to 5
    print i
next i


print "After the for loop i = " + i
```

The above code will display the following output.

```
1
2
3
4
5
After the for loop i = 6
```

Alternative format

The `for` loop also allows us to increment or decrement the index `variable` by steps other than the default `+1`, using the alternative format shown below. The **step** keyword is used to specify the value `step_expression` by which the index `variable` is incremented or decremented after each iteration.

```
for variable = start_expression to stop_expression step
step_expression
   statements
next variable
```

Example

In the following example, the first `for` loop increments the variable `i` by `2` in each iteration whereas the second `for` loop decrements the variable `j` by `2` in each iteration.

```
for i = 1 to 10 step 2
    print "i = " + i
next i


print "After the for loop i = " + i


for j = 10 to 1 step -2
    print "j = " + j
next j


print "After the for loop j = " + j
```

The above code will display the following output.

```
1
3
5
7
9
After the for loop i = 11
10
8
6
4
2
After the for loop j = 0
```

Function

Format

```
function function_name ( function_variable_list )
(tab)statement(s)
end function
```

Description

The `function` statement creates a reusable block of code that receives zero or more arguments (i.e. values), processes those arguments, and optionally returns a value. Strings, integers, and floating point numbers may be returned by a function by executing the `return` statement with a value (or by assigning the name of the function a value and allowing the `end function` statement to be executed).

All variables used within the function will be local to the function and will not change the values in the calling code.

Function variables may be a list of zero or more, comma separated, variables.

Arrays and variables may be passed by reference using the `ref` function.

Functions can be defined anywhere in your program, and cannot be defined within another function, `Subroutine` or control block (`If/Then, Do/Until, …`)

Example

```
print double("Hello")
print double(9)
print triple(3)
end

function double(a)
   double = a + a
end function

function triple(b)
   return b * 3
end function
```

will display

```
HelloHello
18
9
```

Goto
Format

```
goto label
...
label: statement
```

Description

Jumps to the statement at the specified `label` and continues executing from the labeled statement. Any statement can be begin with a `label` followed by a colon. Labels can be used as destinations for **goto**, **gosub**, and **onerror** statements.

Example

The following program has two labels: `start` and `exit`. If the user types "Hello", then the program jumps to the statement labeled `exit`, else it jumps to the statement labeled `start`.

```
start: input "Say Hello: ", message$
if (message$ = "Hello") then
    goto exit
else
    goto start
end if
exit: print "Bye!"
```

The above program will work as follows

```
Say Hello: Ni Hao
Say Hello: Namaskar
Say Hello: Vanakkam
Say Hello: Hello
Bye!
```

Did you know? Goto is considered harmful!

Novice programmers often find the `goto` statement very convenient for writing simple short programs. However, you may be surprised that `goto` statements are strongly discouraged. As a program grows in complexity, `goto` statements lead to undisciplined control flow structure, which makes larger program extremely hard to debug and maintain. Whenever you find yourself wanting to use the `goto` statement, there is almost always a better way to restructure your code using one of the other control flow statements, such as `if/then/else`, `for/next`, `do/until`, and `while` statements.

Gosub
Input
Onerror
Print
Redim

Return

Format
Description
Example

```
HelloHello
18
9
```

While/Do Loop

Format

```
while boolean_expression
    statements
end while
```

Description

The `while` loop executes the statements within the `while/end while` zero or more times until the boolean expression becomes false. The boolean expression is evaluated each time before the statements are executed.

Example

The following program uses the `while` loop to print a message as many times as the user specifies.

```
input "How many Hellos? ", howmany
index = 1
while (index <= howmany)
    print "Hello " + index
    index = index + 1
end while
print "Bye!"
```

The above program will work as follows.

```
How many Hellos? 5
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Bye!
```

Learn More: Continue while

Format

```
while boolean_expression
   statements
   continue while
   statements
end while
```

Description

The `continue while` statement forces the program to skip over rest of the statements within a `while` loop and test the `boolean_expression`.

Example

The following program greets all names except "Mickey" until the name "Foo" is entered.

```
name$ = ""
while (name$ <> "Foo")
   input "Who? ", name$
   if (name$ = "Mickey")then
     continue while
   end if
   print "Hello " + name$
end while
print "Bye!"
```

Learn More: Exit while

Format

```
while boolean_expression
   statements
   exit while
   statements
end while
```

Description

The `exit while` statement forces the program to exit the `while` loop.

Example

The following program greets all names until the name "Foo" is entered. If the name "Mickey" is entered, then the program exits the `while` loop without greeting "Mickey".

```
name$ = ""
while (name$ <> "Foo")
   input "Who? ", name$
   if (name$ = "Mickey")then
     exit while
   end if
   print "Hello " + name$
end while
print "Bye!"
```

Template

Format

```
HelloHello
18
9
```

Description
Example

```
HelloHello
18
9
```

See Also

Data Types

Operators

Expressions

Work in progress…

Expressions are one of the fundamental concepts in any programming language. An Expression is a combination of values and operators that evaluate to a result.

For example, 1 + 2 - 3 is an expression of three constants and two operators that evaluates to -1. Likewise a + b is an expression of two variables that evaluates to the sum of whatever values a and b hold.

Consider the print statement below, which takes an expression and prints it as a String.

```
print "Number " a + " is greater than " + b
```

So this is a good time to look at the page for Expressions in Basic256 (which is unfortunately incomplete). So, instead we look at the page for Operators in Basic256 which has tons of interesting information that I encourage you to read. Particularly, the section on String Operators is what we are searching for.

It tells us that there are actually at least three ways of concatenating two or more expressions into a string expressions, namely, (a;b) , (a + b), and (a & b). In the print statement above, I chose the + operator for concatenation, because it is more intuitive to me (though it can also be confusing if the two operands are numbers).

If you are curious, the fourth expression in that table of String operators is also interesting: a * i concatenates a to itself i times, like in "Hello!" * 4.

Another interesting fact to note in the print statement above is that we are mixing up Strings and Numbers into one expression using the + operator. Specifically, `"Variable "` and `" is greater than "` are strings, whereas `a` and `b` are numbers. Now mixing different data types like this may seem intuitive for this print statement. However in some other programming languages (such as C), mixing data types is either disallowed or strongly frowned upon. That's because by mixing data types, programmers get sloppy and make mistakes. But programming languages for beginners, like Basic and Python, allow you to mix data types because it is more intuitive. Just something to keep in mind as you learn more advanced proigramming languages later.

You can read more about Data Types here, which is another fundamental concept in programming languages.

In Basic language, although data types are not explicitly declared for convenience, you can find out the type of a variable or an expression using the TypeOf function. Try it!

Functions

Explode

Format

```
variable = explode (string_expression, delimiter_expression )
variable = explode (string_expression, delimiter_expression, boolean_expression)
```

returns a list of strings. Typically this function is used to create an array.

Description

Splits up the `string_expression` into substrings wherever the `delimiter_expression` occurs.

The optional `boolean_expression` specifies whether the search will treat upper and lower case letters the same.

Example 1

```
# explode on spaces
a$ = "We all live in a yellow submarine."
w$ = explode(a$," ")
for t = 0 to w$[?]-1
   print w$[t]
next t
```

will display

```
We
all
live
in
a
yellow
submarine.
```

Example 2

```
# explode on A or a
a$ = "All_around_An_almond_mountain."
w$ = explode(a$,"A",true)
for t = 0 to w$[?]-1
   print w$[t]
next t
```

will display

```
ll_
round_
n_
lmond_mount
in.
```

Example 3

```
# explode on a comma
a$ = "1,2,3,77,foo,9.987,6.45"
n = explode(a$, ",")
for t = 0 to w$[?]-1
   print n[t]
next t
```

will display

```
1
2
3
77
foo
9.987
6.45
```

Implode
Format

```
implode ( variable[] )
implode ( variable[] , delimiter_expression )
implode ( variable[] , row_delimiter_expression, column_delimiter_expression )
implode ( { x1, y1, x2, y2, x3, y3 ... } )
implode ( { x1, y1, x2, y2, x3, y3 ... } , delimiter_expression )
implode ( { x1, y1, x2, y2, x3, y3 ... } , row_delimiter_expression,
column_delimiter_expression )
implode ( { {x1, y1}, {x2, y2}, {x3, y3} ... } )
implode ( { {x1, y1}, {x2, y2}, {x3, y3} ... } , delimiter_expression )
implode ( { {x1, y1}, {x2, y2}, {x3, y3} ... } , row_delimiter_expression,
column_delimiter_expression )
```

returns `string_expression.`

Description

Append the elements in an array into a string. Optionally placing the `delimiter_expression` between the elements. This is functionally the opposite of the `Explode` function.

Example

```
dim a$(1)
dim n(1)

a$ = explode("How now brown cow"," ")
b$ = implode(a$[],"-")
print b$
c$ = implode(a$[])
print c$

n = explode("1,2,3.33,4.44,5.55",",")
n1$ = implode(n[],", ")
print n1$
n2$ = print implode(b[])
print n2$
```

will display

```
How-now-brown-cow
Hownowbrowncow
1, 2, 3.33, 4.44, 5.55
123.334.445.55
```

Left

Format

```
string2$ = left ( string1$, n)
```

Description

The left function takes two arguments, a string string1$ and a number n, and returns another string string2$ that consists of the first n characters of string1$.

Example

```
s0$ = "Hello"
s1$ = left( s0$, 1)
print s1$
s2$ = left( s0$, 2)
print s2$
s3$ = left( s0$, 3)
print s3$
s4$ = left( s0$, 4)
print s4$
s5$ = left( s0$, 5)
print s5$
```

will display

```
H
He
Hel
Hell
Hello
```

Length

Format

```
n = length ( string$ )
```

Description

The `length` function returns the number of characters `n` in the string argument `string$`.

Example

```
s$ = "Hello"
len = length(s$)
print len
print right(s$, len - 1)
```

will display

```
5
ello
```

Right

Format

```
string2$ = right ( string1$, n)
```

Description

The `right` function takes two arguments, a string `string1$` and a number n, and returns another string `string2$` that consists of the last n characters of `string1$`.

Example

```
s0$ = "Hello"
s1$ = right( s0$, 1)
print s1$
s2$ = right( s0$, 2)
print s2$
s3$ = right( s0$, 3)
print s3$
s4$ = right( s0$, 4)
print s4$
s5$ = right( s0$, 5)
print s5$
```

will display

```
o
lo
llo
ello
Hello
```

Template

Format
Description
Example

```
HelloHello
18
9
```

Data Structures

Arrays

Description

An array is a list of values that have a common name. Each value in an array in identified by an index. You can think of an arrays as many values arranged in a single row, like this.

Value[0]	Value[1]	Value[2]

Arrays can also be two dimensional, meaning that the values are arranged in rows and columns, like this

Value[0][0]	Value[0][1]	Value[0][2]
Value[1][0]	Value[1][1]	Value[1][2]

Arrays are allocated using the `dim` command or re-sized using `redim`. They may hold numeric or string data. For example, the following code creates an array of numbers called `a` and fills them up with three numbers.

```
dim a(3)
a[0] = 9
a[1] = 99
a[2] = 999
for i = 0 to 2
   print a[i]
next i
```

9	99	999

After you create an array, you can access the individual values of the array as follows:

- For a one dimensional array, the value at position `index` is accessed using square braces, as in `array[index]`. For example, `a[0]` accesses the first value of the array `a` and `a[10]` accesses its 11th value.
- For a two-dimensional array, the values are accessed by specifying the `row` and `column` number of the element, as in `array[row][column]`. For example, `a[0][2]` accesses the array value at row 0 and column 2.
- By default arrays are indexed using a number in the range of 0 to array_length-1. You may optionally change the array index to a range of 1 to array_length by using the `ArrayBase` statement.

Array lengths may also be extracted using `[?]` `[?,]` and `[,?]` on the end of the array variable.

- `[?]` returns the length of a one-dimensional array.
- `[?,]` or `[?][]` return the number of rows of a two-dimensional array.
- `[,?]` or `[][?]` return the number of columns of a two-dimensional array.

ArrayLength

Format
```
variable [?]
variable [?,]
variable [?][]
variable [,?]
variable [][?]
```

Array lengths may be extracted using `[?]` `[?,]` and `[,?]` on the end of the array variable.
- `one_d[?]` returns the length of a one-dimensional array called `one_d`.
- `two_d[?,]` or `two_d[?][]` returns the number of rows of a two-dimensional array called `two_d`.
- `two_d[,?]` or `two_d[][?]` returns the number of columns of a two-dimensional array called `two_d`.

Assigning values to an array

Values may be assigned to an array in one of five ways:
1. By using the `dim` statement to reserve space for the array in the computer's memory and then assigning each individual element.

```
dim a(10)
for t = 0 to a[?]-1
    a[t] = t*t
    print a[t]
next t
```

2. By using a list to create and assign an array.

```
a = {{0,1,2},{3,4,5},{6,7,8}}
b[] = {1,2,3,4}
for i = 0 to a[?][]-1
    for j = 0 to a[][?]-1
        print a[i][j]
    next j
next i
for i = 0 to b[?]-1
    print b[i]
next i
```

3. By using the `dim` statement to copy an existing array into another array.

```
a = {1,2,3,4}
dim b = a[]
```

4. By using the `Explode` or `Explodex` functions to split a string into an array.

```
a$ = explode("how now brown cow"," ")
for i = 0 to a$[?]-1
    print a$[i]
next i
```

5. Using the `fill` assignment operator (with or without `dim`)

```
dim c fill "stuff"
dim e[] fill 0
b fill ""
a[] fill -1
```

List

Challenge Problems

For Loop and String Input

Learning Exercise

First, learn the following concepts:
- For loop
- Input string

Programming Problem

Write a program that asks the user for a message and the number of times to print the message. Then it prints the message as many times as the user told it to print. Each printed message is preceded by a count of the number of times it has printed so far.

Example

```
What message do you want to print? Hello World!
How many times do you want to print your message? 5
1. Hello World!
2. Hello World!
3. Hello World!
4. Hello World!
5. Hello World!
```

Bonus Problem

Print the numbers before the message in descending order.

Example

```
What message do you want to print? Hello World!
How many times do you want to print your message? 5
5. Hello World!
4. Hello World!
3. Hello World!
2. Hello World!
1. Hello World!
```

Larger or Smaller or Equal?

Learning Exercise

First learn the following concepts:

- `If/then/else` statement
- `Input` statement
- `Print` statement

Programming Problem

Write a program that asks the user to input two numbers. Then it prints whether the first number is larger than, smaller than, or equal to the second number.

Example 1

```
Input the first number: 5
Input the second number: 6
5 is smaller than 6
```

Example 2

```
Input the first number: 20
Input the second number: 10
20 is larger than 10
```

Example 3

```
Input the first number: 99
Input the second number: 99
The two numbers are equal
```

Largest, Smallest, and Running Sum

Learning Exercise

First, learn how to use the following concepts:
- `goto` statement
- `while` loop
- `do/until` loop

Programming Problem

Write a program that keeps asking the user to input a number and prints the largest, smallest, and sum of the numbers read so far, until the user enters -9999.

Write three different programs to solve this problem
- Program 1: using `goto` statement
- Program 2: using `while` loop
- Program 3: using `do/until` loop

Example

```
Input a number: 10
Largest number read so far is 10
Smallest number read so far is 10
Sum of numbers read so far is 10


Input a number: 5
Largest number read so far is 10
Smallest number read so far is 5
Sum of numbers read so far is 15


Input a number: -1
Largest number read so far is 10
Smallest number read so far is -1
Sum of numbers read so far is 14


Input a number: -9999
Bye!
```

Pig Latin Word

Learning Exercise

First, learn to use the following concepts:

• `do/until` statement
• `length()` function
• `left()` function
• `right()` function

Programming Problem

Write a program that asks the user to input a word and translates the word into Pig Latin.

A Pig Latin translation of an English word is as follows: Take the first letter of the word, move it to the end of the word, and append the letter 'a' at the end. For example, the Pig Latin translation for the word "orange" will be "rangeoa", "banana" will be "ananaba", and so on. When the user types "bye", the program ends with the message "yeba".

Example

```
Type a word: orange
Pig Latin: rangeoa
Type a word: apple
Pig Latin: ppleaa
Type a word: latin
Pig Latin:  atinla
Type a word: bye
yeba
```

Bonus Problem

Translate words input by the user to Cow Latin as follows: Take the last letter of each word, move it to the beginning of the word, and append the string 'oo' to the end of the word. For example, "orange" translates to "eorangoo", "apple" translates to "eapploo", and "bye" to "ebyoo.

Pig Latin Sentence

Learning Exercise

First, learn how to use the following concepts:

• Arrays

• Explode function

• Implode function

Programming Problem

Write a program that asks the user to input a sentence. Then it prints a new sentence in which each word or the original sentence is translated into Pig Latin. A Pig Latin translation of an English word is as follows: Take the first letter of the word, move it to the end of the word, and append the letter 'a' at the end. For example, the Pig Latin translation for the word "orange" will be "rangeoa", "banana" will be "ananaba", and so on. When the user types "bye bye", the program ends with the message "yeba yeba".

Example

Input a sentence: I saw a pig on a tree

Pig Latin: Ia awsa aa igpa noa aa reeta

Input a sentence: cats rained down from the sky

Pig Latin: atsca ainedra ownda romfa heta kysa

Input a sentence: bye bye

yeba yeba

Bonus Problem

Translate sentences input by the user so that each word of a sentence is translated to Cow Latin as follows: Take the last letter of each word, move it to the beginning of the word, and append the string 'oo' to the end of the word. For example, "orange" translates to "eorangoo", "apple" translates to "eapploo", and "bye" to "ebyoo.

Example

Input a sentence: I saw a pig on a tree

Cow Latin: Ioo wsaoo aoo gpioo nooo aoo etreoo

Input a sentence: cats rained down from the sky

Cow Latin: scatoo draineoo ndowoo mfrooo ethoo yskoo

Input a sentence: bye bye

ebyoo ebyoo

Calculator for two numbers

Learning Exercise

First, learn how to use the following concepts:

- `Explode` function
- `while` loop or `do/until` loop
- `begin case/case/end case`
- `int` function to convert a string expression to a number

Programming Problem

Write a program that behaves like a simple calculator that does addition. subtraction, multiplication, or division on two numbers. The program terminates once the user types "End".

Example

```
Input: 3 + 2
Answer: 5
Input: 3 – 2
Answer: 1
Input: 3 x 2
Answer: 6
Input: 3 / 2
Answer: 1.5
Input: End
Bye!
```

Prime Number

Learning Exercise

First, learn how to use the following concepts:

- `for` loop
- `if/then/else`
- Modulo operator `mod` or `%` to calculate the remainder
- **Prime Number:** A prime number is a number that is not divisible by any other number except 1 and itself. For example, the number 3 is a prime number because it is not divisible by any other number except 1 and 3. Likewise, 5 and 7 are prime numbers. However, 4 is not a prime number because, besides 1 and 4, it is also divisible by 2. Likewise 6 and 8 are not prime numbers.
- How do we know whether a number N is a prime number? A simple way to tell is to try dividing N by each of the numbers smaller than N. If N is not divisible by any smaller number except 1 and N, then N is prime.
- How do we know whether a number N is divisible by another number M? Calculate the remainder when N is divided by M using integer division. In other words, calculate N modulo M or N%M. If the remainder is *non-zero*, then N is not divisible by M.

Programming Problem

Write a program that asks the user for a number and tells the user whether that number is prime. The program terminates when the user types 0.

Example

```
Input a number: 3
3 is a prime number
Input a number: 9
9 is not a prime number. It is divisible by 3.
Input a number: 6
6 is not a prime number. It is divisible by 2 and 3.
Input a number: 0
Bye!
```

Bonus problem

Ask user for a number N and print all prime numbers up to N.

PEMDAS/BODMAS

Learning Exercise

First, learn how to use the following concepts:

• PEMDAS Rules: https://www.mathsisfun.com/operation-order-pemdas.html

Programming Problem

Write a calculator program that computes an arithmetic expression according to PEMDAS rules. The program terminates

Example

```
Expression: 1 + 2 x 3
Answer: 7
Expression: 6 / ( 5 – 3 ) + 7
Answer: 10
Expression: 4 * (2^3 + 1) – (3 * 4)
Answer: 24
Expression: Bye
Bye!
```

Palindrome

Learning Exercise

First, learn how to use the following concepts:

• A palindrome is a string (word, phrase, sentence) that reads the same forward and backward.

Programming Problem

Write a program that tells you whether or not an input string is a palindrome.

Example

```
Input: Dad
Its a palindrome!
Input: Mom
Its a palindrome!
Input: Boy
Its not a palindrome :(
Input: A man, a plan, a canal, Panama!
Its a palindrome!
Input: Was it a car or a cat I saw?
Its a palindrome!
Input: Abracadabra
Its not a palindrome :(
Input: Abracadacarba
Its a palindrome!
Input: Bye
Bye!
```

Template

Learning Exercise

First, learn how to use the following concepts:

•

Programming Problem

Example

```
HelloHello
18
9
```

Bonus Problem