Tutorial: Loops & Flow Control

This tutorial will involve exercises to get you accustomed to creating and manipulating flow control constructs. It will move on to complex uses of different data structures and comparison operators.

Create a *program* called `isthenumberbig.pro`. As you might infer from the name, it will tell you if a number is big. To create a program, open a file with the name as specified above (`isthenumberbig.pro`) in `gvim`.

This program shall do the following:

1. Read in a number from the user (the `read` statement)
2. Say "That is a big number" if the number is greater than 100 (use an `if` statement)
3. Repeat unless the user has input the number `-1` (so that means this should all be in a `while` loop that checks to make sure the user hasn't input `-1`)

This program should include at least one loop construct; it is up to you to figure out which kind of loop (`while`, `repeat until`, `for`) is most appropriate. It should include an `if` statement. It may require multiple `if` statements and it may require the `break` statement. It may also be useful to have a `case` statement. The `read` statement should have a descriptive prompt. And, as always, there should be comments throughout!

Make sure your code works (test it with a few numbers: -10, 50, 150, 1d30).

stop

Now insert a `stop` statement in your code, somewhere inside the loop. Run the code. You will be dumped out with the following prompt:

```
% Compiled module: $MAIN$.
% Stop encountered: $MAIN$              ##
```

where `##` is the line number of the `stop` statement.

At this point, examine your variables: Try using the `print` and `help` statements. See what's going on.

When you're done, use `.continue` to continue. You'll be stopped again, though, since you're inside a loop. Use `help` and `print` again to see what's changed. Then `.continue` again, then quit. Once you're done examining the inside of your loop, remove the `stop` statement.

crash!

Now, in the same location where you had the `stop` statement, add the following line:
`y = zzzz^2`

This line will cause a crash, with the text:

```
% Variable is undefined: ZZZZ.
% Execution halted at: $MAIN$              15
```

Note that the "crash" has nearly the same effect as the stop statement: you are back in interactive mode, and need to use `.c` to continue. As long as the variable `zzzz` does not exist, you'll continue to be dropped out just like with `stop` above.

However, try defining the number `zzzz=5` (or any number of your choosing; it really doesn't matter). Then do `.c`. What happens next?

BONUS: Comparison from Text

Write a function called `compare_values` in a file `compare_values.pro` and/or `comparison.py` that takes 3 *string* inputs: `value1`, `comparison_name`, and `value2`. The program should use a `case` statement or a series of `if` statements that will return the appropriate comparison. i.e., if you call `print,compare_values(2,'gt',1)`, the function should return the value of `2 gt 1`, which is "True" or `1`.

Make sure it can deal with the following comparison operators:
IDL: `gt lt ge le eq ne < >`
python: `> < >= <= == !=`

Conditionals

Create a new file called `test_conditionals.pro` and/or `test_conditionals.py`. This is a reference file for your own use now and in the future.

You will create a series of "test conditionals" in this file by using the ternary operator. The tests will take the following form:

```
IDL> print,"Is x gt y?  ",(x gt y) ? "yes" : "no"
>>>  print "Is x > y?","yes" if x > y else "no"
```

Create tests for the following, but *fill these out by hand* before you run them! (i.e., circle "yes" or "no")

HINT: To make this easier, you can make use of the `list` and/or the `hash`/`dict` construct. You should use `foreach`/`for` loops. You should definitely make use of the `compare_values` function you created above.

x	comparison		y	yes or no
	IDL	python		
1	gt	>	0	yes no
1	>		0	yes no
5	>		0	yes no
4	>		0	yes no
1	>		7	yes no
5	>		7	yes no
4	>		7	yes no
'0'	eq	==	0	yes no
'1'	eq	==	0	yes no
0.0	eq	==	0	yes no
1.0	eq	==	1	yes no
0.5d	eq	==	0.5	yes no
0.2d	eq	==	0.2	yes no
`2^(-30d)`	eq	==	`2^(-30e)`	yes no
`1.1+2^(-30d)`	eq	==	`1.1+2^(-30e)`	yes no
'yes'	eq / ==	"yes"	yes no	
`sin(!pi)`	lt	<	`sin(!dpi)`	yes no
`cos(!pi)`	eq	==	`cos(!dpi)`	yes no

Add any tests you can think of to this.

Once you have completed this exercise, `git add` the file, `git commit -a` to commit the changes, and `git push` them to your repository.