LING/C SC/PSYC 438/538

Lecture 5

Sandiway Fong

Today's Topics

- A remark on preposition senses
- Quoting rules for PowerShell and the Bash Shell (*use as a reference*)
- Perlintro
- Homework 5 out today

No class next week

I'll be in Osaka, Japan for two weeks

- Please work through `perlintro` (*and try the examples*).
- And you have Homework 5 to do (*after this class*).
- Plan is we **not** have lectures on 12th and 14th September
  - Homework deadline shifted
- Following week:
  - **TBA:** Monday 19th: or Wednesday 21st
    - let's meet on Zoom to review Homework 5 and related material

Word senses: Prepositions

	# of senses
Of	12
With	11
By	11
For	10
On	10
To	8
At	6
About	6
Into	6
Between	6
Out	6
In	5
Like	5
Through	5
Over	5
Off	5
From	3
As	3
After	3

- Georgia West compiled this! (Thank you)

Shell vs. Programming Language

From last time, historic conflict over quoting behavior (' ").

- On the command line:
  - the Terminal (Shell) gets first dibs, and
  - the programming language, e.g. Perl, gets seconds
- **Choice**:
  - Understand the quoting rules for the Shell, or
  - Write your program always using a plain text file, e.g. `myprog.perl`, run using:

```
perl myprog.perl
```

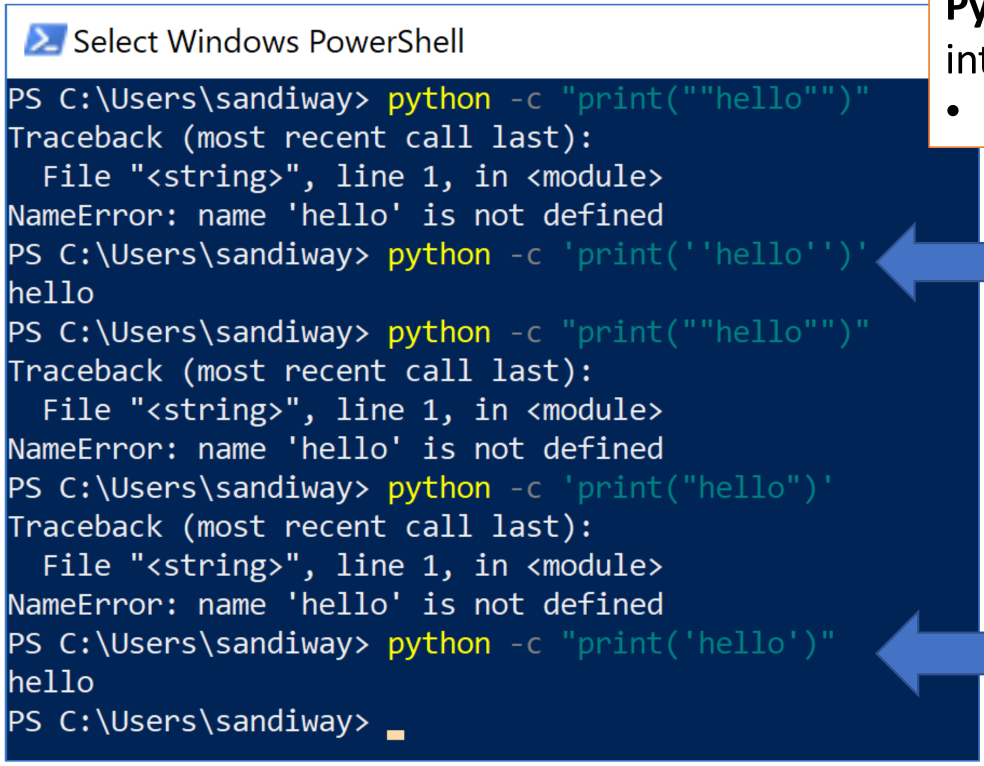
  - advantage: you don't have to worry about the Shell quoting rules

Windows PowerShell

- Quoting rules are here:
  - https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_quoting_rules?view=powershell-7
  - double quotes (") **allow** interpolation
  - single quotes (') means pass string unchanged (*but see 2. below*)

  1. but PowerShell doesn't use the standard backslash (\) for escaping, uses backtick (`) instead
  2. can also double up, e.g. '' = '+' (*two single quotes*) inside a single quoted string

Windows PowerShell and Python


Select Windows PowerShell

```
PS C:\Users\sandiway> python -c "print(""hello"")"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NameError: name 'hello' is not defined
PS C:\Users\sandiway> python -c 'print(''hello'')'
hello
PS C:\Users\sandiway> python -c "print(""hello"")"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NameError: name 'hello' is not defined
PS C:\Users\sandiway> python -c 'print("hello")'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
NameError: name 'hello' is not defined
PS C:\Users\sandiway> python -c "print('hello')"
hello
PS C:\Users\sandiway> _
```

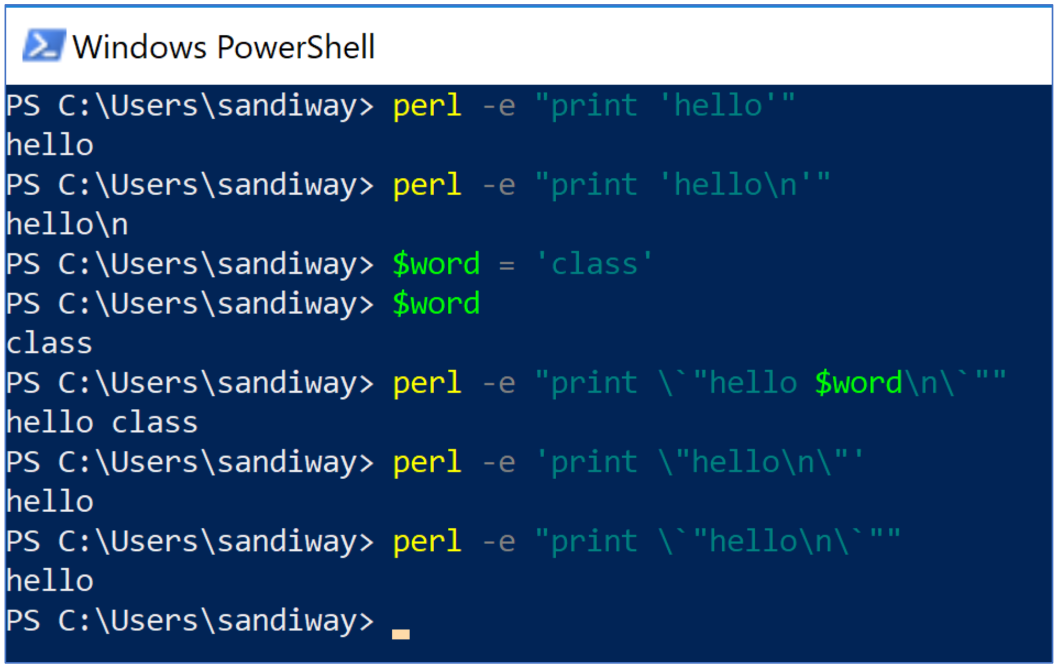
Python uses single and double quotes interchangeably to delimit strings.

- Unquoted string is a variable name (or keyword)

doubled single quotes inside single-quoted string

single quotes inside double-quoted string

Windows PowerShell and Perl



Windows PowerShell

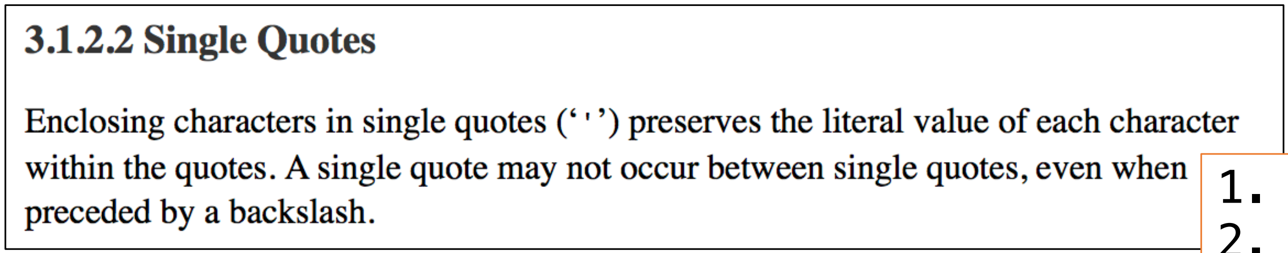
```
PS C:\Users\sandiway> perl -e "print 'hello'"
hello
PS C:\Users\sandiway> perl -e "print 'hello\n'"
hello\n
PS C:\Users\sandiway> $word = 'class'
PS C:\Users\sandiway> $word
class
PS C:\Users\sandiway> perl -e "print \`"hello $word\n\`""
hello class
PS C:\Users\sandiway> perl -e 'print \"hello\n\"'
hello
PS C:\Users\sandiway> perl -e "print \`"hello\n\`""
hello
PS C:\Users\sandiway> _
```

Perl is quirky on Windows:

- " needs to be \"
- Inside single quotes, \" is ok to Perl
- Inside double quotes, needs to be \`"

Bash Shell quoting

- Bash shell (MacOS, Linux):
  - manual: http://www.gnu.org/software/bash/manual/



3.1.2.2 Single Quotes

Enclosing characters in single quotes (‘'’) preserves the literal value of each character within the quotes. A single quote may not occur between single quotes, even when preceded by a backslash.

1. `'  …  '` *fine if no ' inside*
2. `'  …  '  …  '  …  '` *doesn't work*
3. `'   … \'  … \'  …'` *cannot work*

want this (@a is an array):

```
@a=('a', 'b', 'c')
```

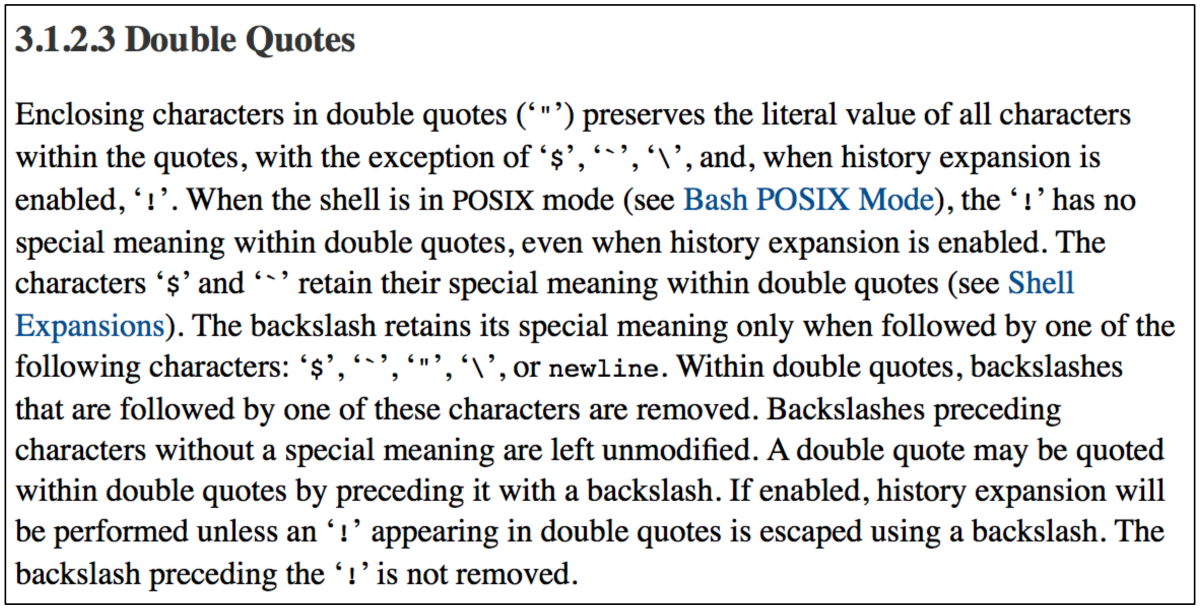
but we can't write:

```
perl –e '@a=(\'a\',\'b\',\'c\'); print "@a\n"'
```

so what can we do? (*see next slide*)

Bash Shell quoting

- Bash shell (MacOS, Linux):



3.1.2.3 Double Quotes

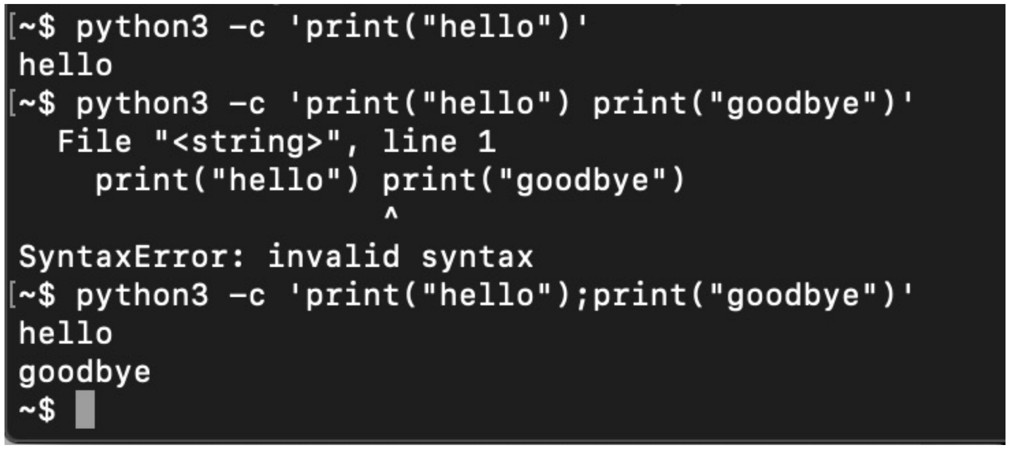
Enclosing characters in double quotes (‘`"`’) preserves the literal value of all characters within the quotes, with the exception of ‘`$`’, ‘`` ` ``’, ‘`\`’, and, when history expansion is enabled, ‘`!`’. When the shell is in POSIX mode (see Bash POSIX Mode), the ‘`!`’ has no special meaning within double quotes, even when history expansion is enabled. The characters ‘`$`’ and ‘`` ` ``’ retain their special meaning within double quotes (see Shell Expansions). The backslash retains its special meaning only when followed by one of the following characters: ‘`$`’, ‘`` ` ``’, ‘`"`’, ‘`\`’, or `newline`. Within double quotes, backslashes that are followed by one of these characters are removed. Backslashes preceding characters without a special meaning are left unmodified. A double quote may be quoted within double quotes by preceding it with a backslash. If enabled, history expansion will be performed unless an ‘`!`’ appearing in double quotes is escaped using a backslash. The backslash preceding the ‘`!`’ is not removed.

can write this (*but not elegant*):

```
perl -e "@a=(\"a\",\"b\",\"c\"); print \"@a\n\""
```

; separator

- Even in Python, you might need ; sometimes.

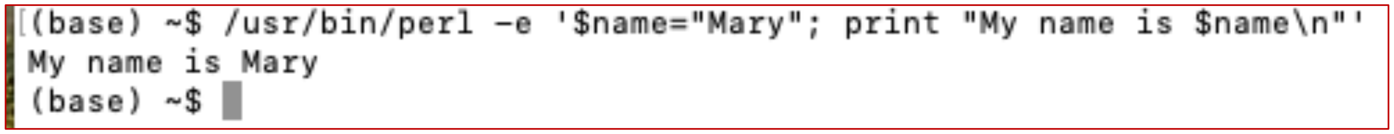


```
[~$ python3 -c 'print("hello")'
hello
[~$ python3 -c 'print("hello") print("goodbye")'
  File "<string>", line 1
    print("hello") print("goodbye")
                       ^
SyntaxError: invalid syntax
[~$ python3 -c 'print("hello");print("goodbye")'
hello
goodbye
~$
```

- In Python:
  - **visible punctuation is not generally used**
  - newlines separate statements
  - indentation (tabs) matter
- Most other programming languages:
  - use ;
  - { … } grouping

Perl vs. Python3

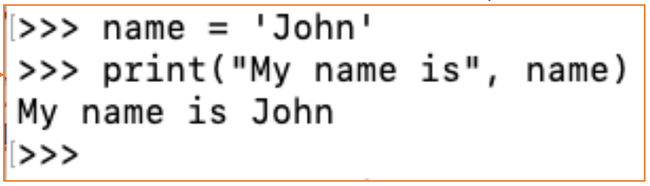
Variable interpolation



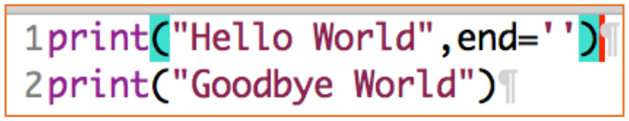
```
(base) ~$ /usr/bin/perl -e '$name="Mary"; print "My name is $name\n"'
My name is Mary
(base) ~$
```

- Perl:
  1. `print "My name is $name\n";`
  2. `print "My name is \$name\n";`
  3. `print 'My name is \$name\n';`
- Python3:
  - `print("My name is", name)`
- Python2.7:
  - `print("My name is", name)`

Note: additional white spacing



```
>>> name = 'John'
>>> print("My name is", name)
My name is John
>>>
```



```
print("Hello World",end='')
print("Goodbye World")
```

perlintro

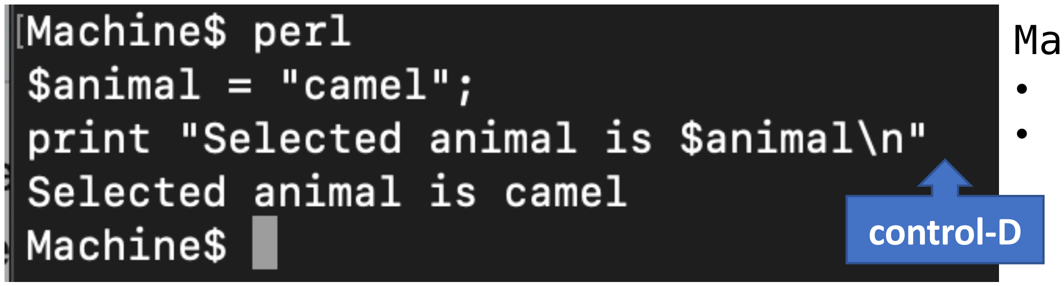
Scalars:

- variable names begin with **$** - **note**: *no such type naming requirement in Python*
- basically single unit items

1. **Numbers** (integer, floating point)
   - that take up one 32/64 bit word of memory
   - Python includes complex numbers (`cmath` library)
     - also user-definable precision floats (`decimal` library)
2. **Strings** (double ".." or single quoted '..')
3. **References** (pointers to (non-)scalars)

perlintro

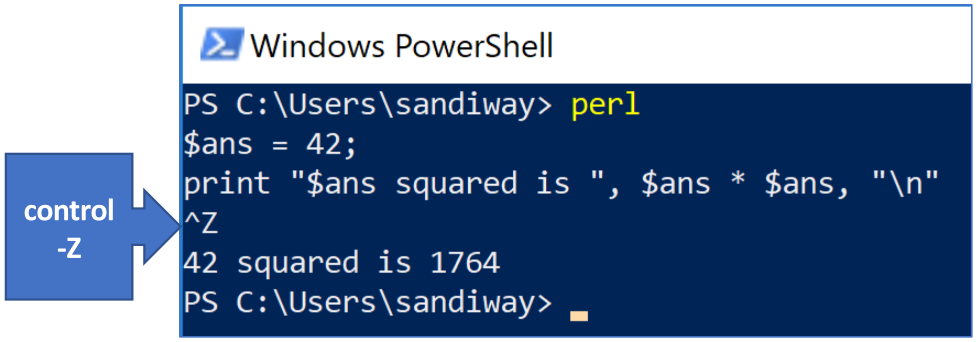
- https://perldoc.perl.org/perlintro.html
- Please read the Scalars ($) section …



```
Machine$ perl
$animal = "camel";
print "Selected animal is $animal\n"
Selected animal is camel
Machine$
```

`Machine$` is my prompt, don't type that!

- I am using the terminal as the file input to Perl
- Type **control-D** (EOF = End Of File) to send to Perl.



```
PS C:\Users\sandiway> perl
$ans = 42;
print "$ans squared is ", $ans * $ans, "\n"
^Z
42 squared is 1764
PS C:\Users\sandiway>
```

`PS C:\Users\sandiway>` is my prompt, don't type that!

- I am using the terminal as the file input to Perl
- Type **control-Z** RETURN (EOF) to send to Perl.

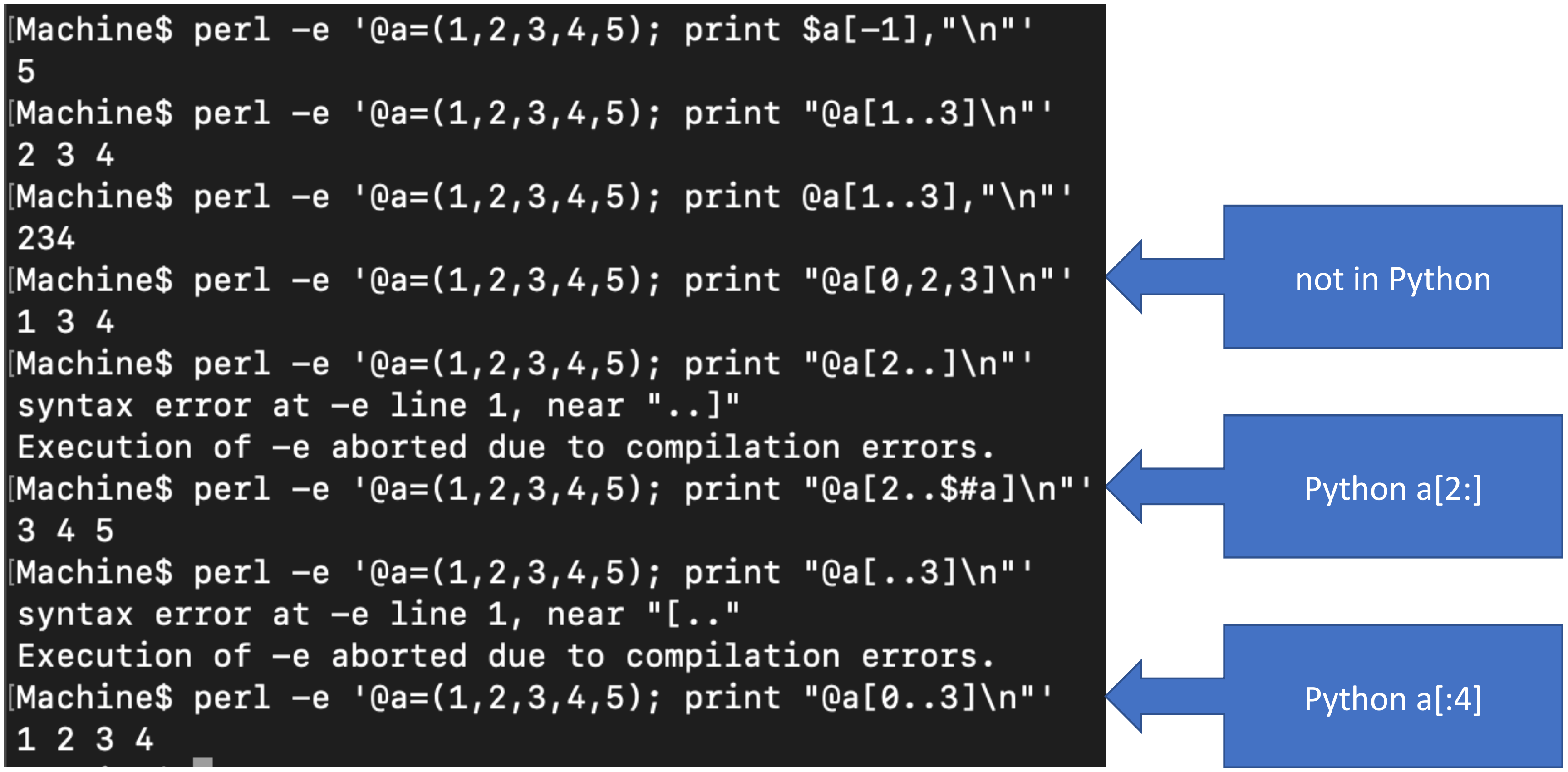
perlintro

Non-scalar data type: **array**

- prefix with @, array is `@name` (name = array name)
- indexed from 0
- `$name[index]`, an element of the @name array (**notice scalar** $)
- $#name, index of last element
- `print "@name"` (spaces inserted), print @name (no spaces)

controlled by system variable $"
default value: a space

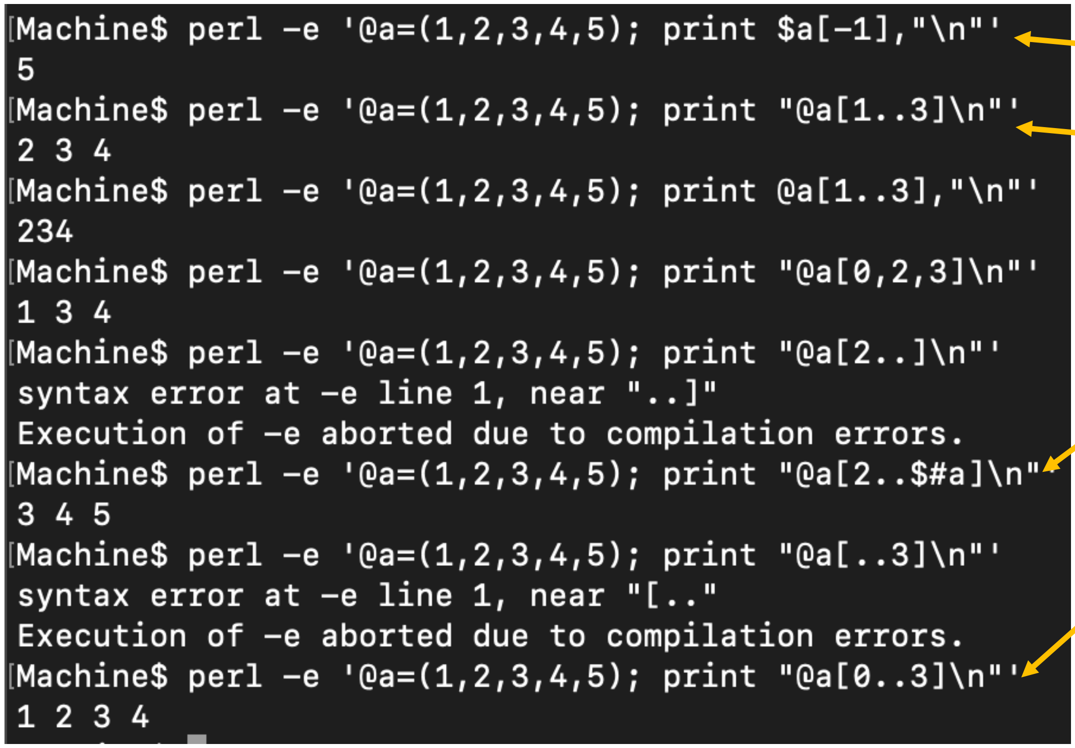
perlintro



```
[Machine$ perl -e '@a=(1,2,3,4,5); print $a[-1],"\n"'
5
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[1..3]\n"'
2 3 4
[Machine$ perl -e '@a=(1,2,3,4,5); print @a[1..3],"\n"'
234
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[0,2,3]\n"'
1 3 4
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[2..]\n"'
syntax error at -e line 1, near "..]"
Execution of -e aborted due to compilation errors.
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[2..$#a]\n"'
3 4 5
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[..3]\n"'
syntax error at -e line 1, near "[.."
Execution of -e aborted due to compilation errors.
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[0..3]\n"'
1 2 3 4
```

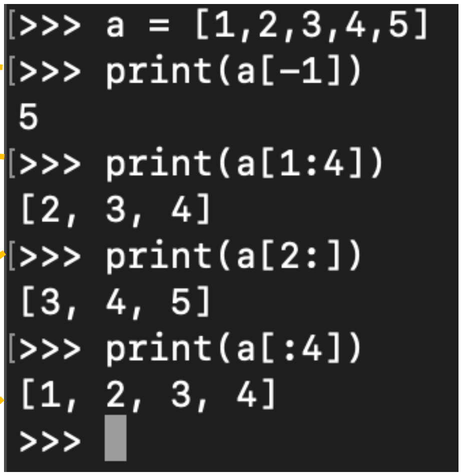
perlintro

- Perl



```
[Machine$ perl -e '@a=(1,2,3,4,5); print $a[-1],"\n"'
5
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[1..3]\n"'
2 3 4
[Machine$ perl -e '@a=(1,2,3,4,5); print @a[1..3],"\n"'
234
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[0,2,3]\n"'
1 3 4
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[2..]\n"'
syntax error at -e line 1, near "..]"
Execution of -e aborted due to compilation errors.
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[2..$#a]\n"'
3 4 5
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[..3]\n"'
syntax error at -e line 1, near "[.."
Execution of -e aborted due to compilation errors.
[Machine$ perl -e '@a=(1,2,3,4,5); print "@a[0..3]\n"'
1 2 3 4
```

- Python



```
[>>> a = [1,2,3,4,5]
[>>> print(a[-1])
5
[>>> print(a[1:4])
[2, 3, 4]
[>>> print(a[2:])
[3, 4, 5]
[>>> print(a[:4])
[1, 2, 3, 4]
>>>
```

perlintro

Notes from the tutorial:

- semicolon (;) is not always necessary
  - **Command separator** semantics vs. end of command (termination) token
  - **Best practice**? Terminate every command with a semicolon
- Variable types:
  - Every variable type has its own namespace. (cf. Python)
  - This means that $foo and @foo are two different variables.
  - It also means that $foo[1] is a part of @foo, not a part of $foo. **This may seem a bit weird, but that's okay, because it is weird.**

Perl Arrays

like a simple ordered list… (in **Python**, we use a list/sequence)

- Literal:
  - *@ARRAY* = (… , … , …)   (round brackets; comma separator)
  - **Python**: array = [… , … , …]
- Access:
  - $*ARRAY*[*INDEX*]   (zero-indexed; negative indices ok; slices ok)
  - **Python**: array[index]
- Index of last element:
  - $#array   (a scalar)
- Last element
  - $array[$#array] or $array[-1]
  - **Python**: array[-1]
- Slice of an array
  - @array[i..j]   (i,j indices)
  - **Python**: array[i:j]   (i and/or j can be left off, also a step can be included, e.g. ::-1
- Coercion
  - *@ARRAY*   = number of elements in scalar context
  - **Python**: len(array)

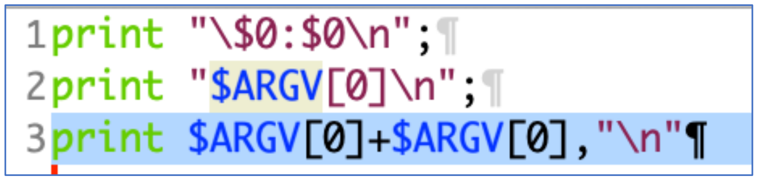
Perl Arrays

- Built-in arrays:

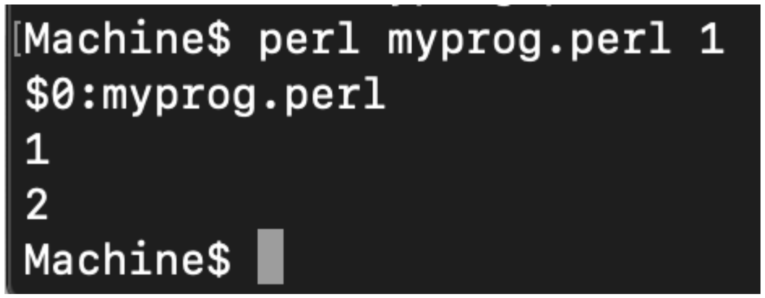
`@ARGV`	(command line arguments; coercion possible)
`$ARGV[0]`	(1st argument)
`$0`	(program name)
`@_`	(sub(routine) arguments)

- **Example**:

myprog.perl



```
print "\$0:$0\n";¶
print "$ARGV[0]\n";¶
print $ARGV[0]+$ARGV[0],"\n"¶
```

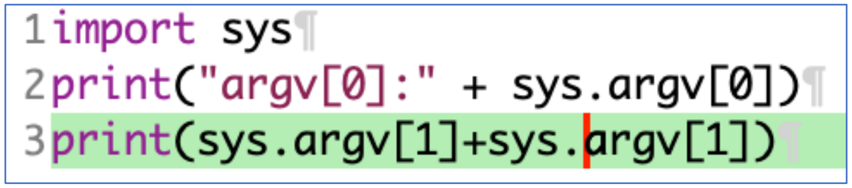


```
[Machine$ perl myprog.perl 1
$0:myprog.perl
1
2
Machine$
```

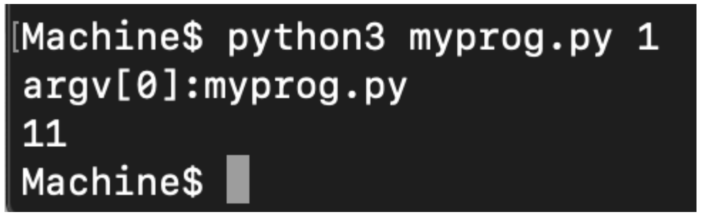
Perl Arrays

- Python argv:
  - `import sys`
  - `sys.argv` (list of command arguments as strings)
  - `sys.argv[0]` (Python script name)
  - `sys.argv[1]` (1^{st} argument)
  - **Example**:

`myprog.py`



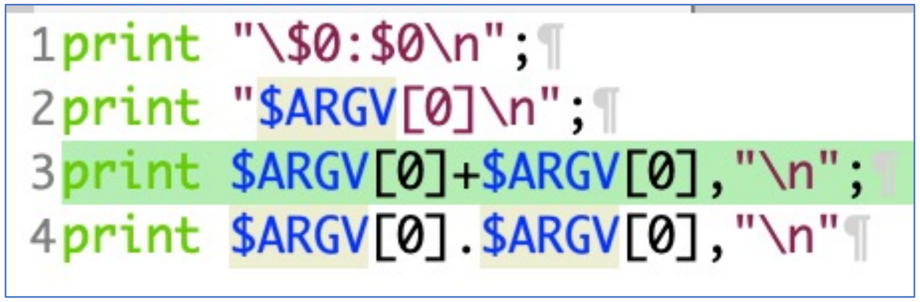
```
import sys
print("argv[0]:" + sys.argv[0])
print(sys.argv[1]+sys.argv[1])
```



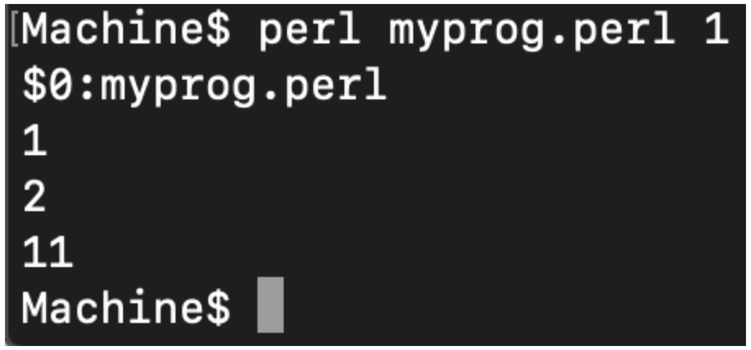
```
Machine$ python3 myprog.py 1
argv[0]:myprog.py
11
Machine$
```

`int(sys.argv[1])` to convert string into an integer

Perl Arrays



```
print "\$0:$0\n";
print "$ARGV[0]\n";
print $ARGV[0]+$ARGV[0],"\n";
print $ARGV[0].$ARGV[0],"\n"
```



```
[Machine$ perl myprog.perl 1
$0:myprog.perl
1
2
11
Machine$ 
```

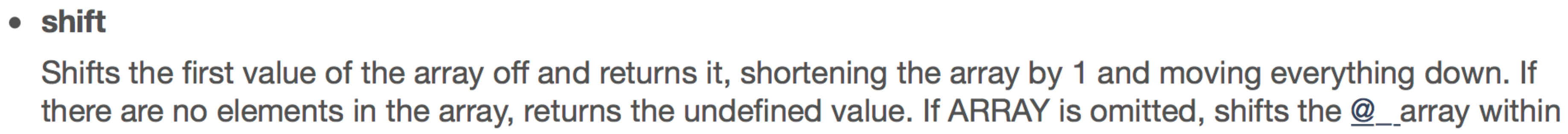
Perl Arrays

- Built-in functions:
  - sort *@ARRAY*; reverse *@ARRAY;*
  - push *@ARRAY, $ELEMENT;* pop *@ARRAY*; (operates at right end of array)
  - shift *@ARRAY*; unshift *@ARRAY, $ELEMENT,* (left end of array)
  - splice *@ARRAY, $OFFSET, $LENGTH, $ELEMENT*
    - *$ELEMENT above can be @ARRAY*
- **Python**:
  - array.sort(), array.reverse()
  - NO push (use array.append() instead), array.pop(),
  - No shift/unshift etc… (but can use slicing and concatenation)

`perlintro:` Perl Arrays

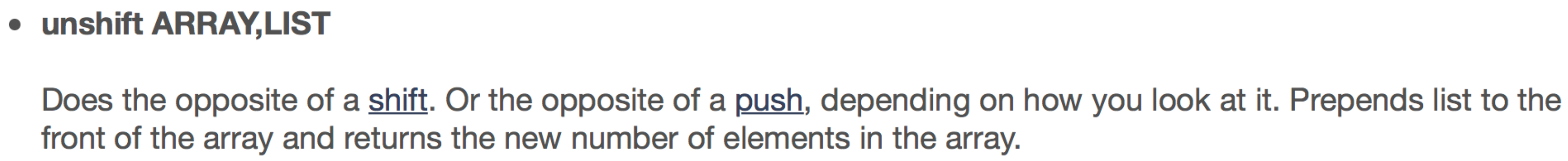
- **shift ARRAY**

Similar to pop/push, but operates at the left end of the array



- **shift**

  Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down. If there are no elements in the array, returns the undefined value. If ARRAY is omitted, shifts the @_ array within



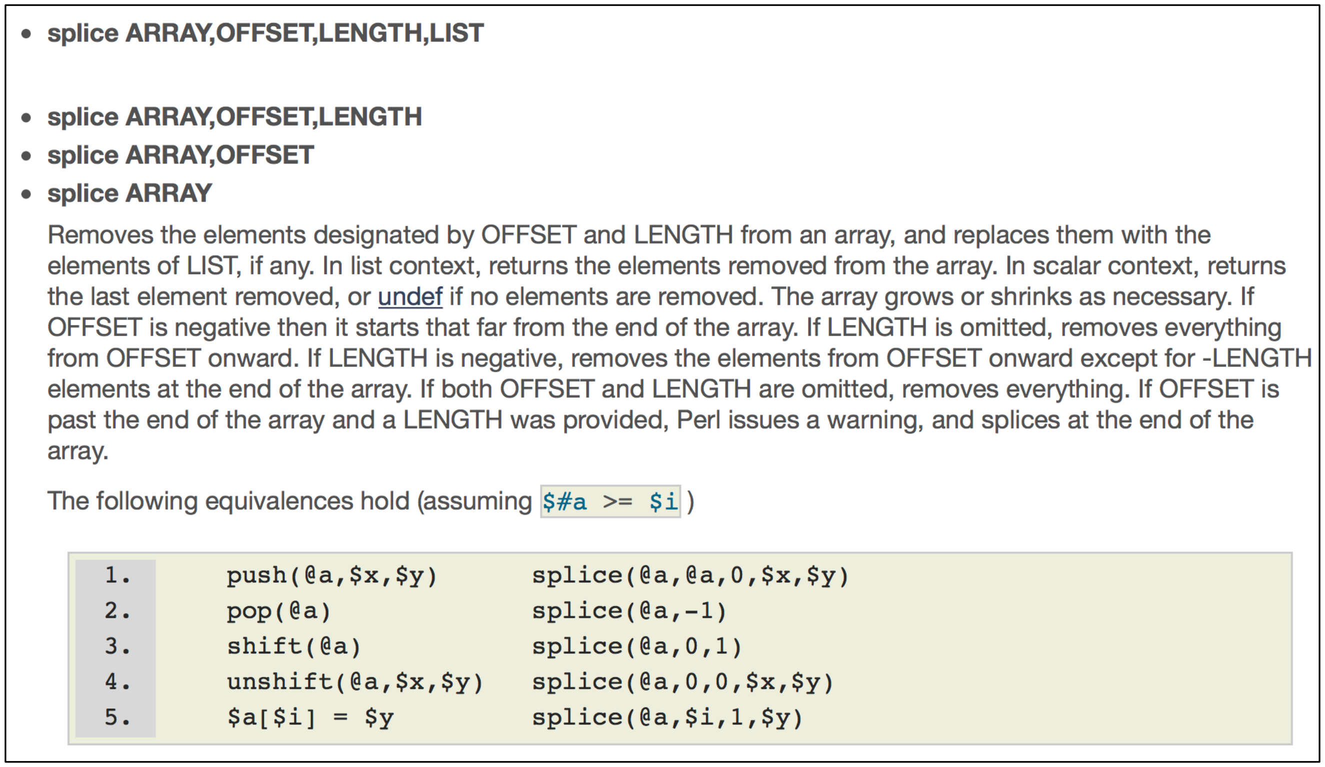
- **unshift ARRAY,LIST**

  Does the opposite of a shift. Or the opposite of a push, depending on how you look at it. Prepends list to the front of the array and returns the new number of elements in the array.

Python doesn't have these defined but can be simulated via slicing and concatenation:

```
array[1:]
list + array
```

`perlintro:` Perl Arrays



- **splice ARRAY,OFFSET,LENGTH,LIST**
- **splice ARRAY,OFFSET,LENGTH**
- **splice ARRAY,OFFSET**
- **splice ARRAY**

Removes the elements designated by OFFSET and LENGTH from an array, and replaces them with the elements of LIST, if any. In list context, returns the elements removed from the array. In scalar context, returns the last element removed, or undef if no elements are removed. The array grows or shrinks as necessary. If OFFSET is negative then it starts that far from the end of the array. If LENGTH is omitted, removes everything from OFFSET onward. If LENGTH is negative, removes the elements from OFFSET onward except for -LENGTH elements at the end of the array. If both OFFSET and LENGTH are omitted, removes everything. If OFFSET is past the end of the array and a LENGTH was provided, Perl issues a warning, and splices at the end of the array.

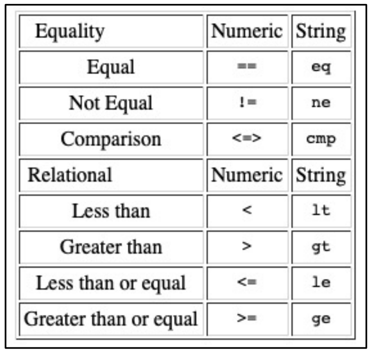
The following equivalences hold (assuming `$#a >= $i`)

```
    push(@a,$x,$y)      splice(@a,@a,0,$x,$y)
    pop(@a)             splice(@a,-1)
    shift(@a)           splice(@a,0,1)
    unshift(@a,$x,$y)   splice(@a,0,0,$x,$y)
    $a[$i] = $y         splice(@a,$i,1,$y)
```

Homework 5: Question 1

Using Perl (not Python)

- Write a program that takes two strings or numbers on the command line and prints out whether they're equal or not, e.g.
  - `perl hw5q1.perl 1 1.0`
  - `equal`
  - `perl hw5q1.perl 1 0.1e1`
  - `equal`
  - `perl hw5q1.perl windy Windy`
  - `not equal`
  - What happens with this? `perl hw5.perl`
- **Resources**:
  - $ARGV[0] and $ARGV[1]
  - if(*comparison*) { *statement* }
    - else { *statement* }
    - elsif (*comparison*) { *statement* }
  - equality (*see table* ⇢)



Equality	Numeric	String
Equal	==	eq
Not Equal	!=	ne
Comparison	<=>	cmp
Relational	Numeric	String
Less than	<	lt
Greater than	>	gt
Less than or equal	<=	le
Greater than or equal	>=	ge

Homework 5: Question 2

- Write a Perl program to sort a list of numbers from the command line and print them in **ascending** order:
  - `perl hw5q2.perl 3 1 4 1 5 9 2 6 5 3 5 8 9 7 9`
  - `1 1 2 3 3 4 5 5 5 6 7 8 9 9 9`
  - `perl hw5q2.perl 20 50 9 5 1`
  - `1 5 9 20 50`
  - `1 20 5 50 9` (*not this*!)

  can you show and explain what happens with?
  - `perl hw5q2.perl a A b B c C`

- **Resources**:
  - @ARGV
  - look up function **sort** and numeric comparison (*not the default case*)
    - https://perldoc.perl.org/functions/sort.html

Homework 5

- Submit to me a PDF with your code and screenshot output:
  - Email: sandiway@arizona.edu
  - Subject: 438/538 Homework 5 **Your Name**
  - Due date: lots of extra time (Sunday 18th midnight)!
  - **Zoom meeting**:
    - review Homework 5 and more
    - Monday 19th 3:30pm Tucson time
    - Tuesday 20th 7:30am Japan time