Python: What's not to like?

Caveat

My favorite languages are
PDP-6 assembler
and
PostScript

Indentation matters

- It's amazingly easy to screw up when editing

```
LIST = []
for p in os.listdir(path) :
  m = HBRE.match(p)
  if m :
    LIST.append(m.group(1))
  LIST.sort()
```

- If it's such a great idea, why not go all the way?

```
x = a+b * c
y = a + b*c
```

Operations return None

- Many cases where the result is obvious, but None is returned instead

`foo.sort()`	should return (the sorted) `foo`
`foo.append(bar)`	should return (the updated) `foo`
`a = b+c`	should return the resulting sum
`a += 1`	should return (the incremented) `a`

Missing operations, requiring extra tests or try/except

- 'Failing' list operations

  `foo.remove(3)` variant could be happy if 3 not in `foo`
  `foo.index(3,x)` could return `x` if 3 not in `foo`
  `foo.pop(4,x)` could return `x` if `len(foo)`$\leq$4

- Divide by 0. and get $\pm$`inf` or `nan` (there should be a way to do this globally)
- In general, there should be a way to handle exceptions, 'fix' them, and continue (which also could be used by a debugger)

Lack of guidance

- Efficiency: It would be nice to know how long various not-so-primitive primitive operations take to execute, and what conditions (e.g., argument types) affect them.
- Memory: It would be nice to know how much space various data types take, and what affects that.
- Best Practices: What's a good way to do *x*?

Debugging Desiderata

- Ability to back up to the point of an exception, fix the problem, and continue
  [checkpoint at the point an exception is raised; if exception bubbles to the top, back up to checkpoint?]
  [may need some finer control if try/except is overused, which sometimes it must be]
- Ability to interrupt a running program, examine and modify variables, execute functions, and resume

Debugging Desiderata (continued)

- Ability to patch code and redefine functions
- Ability to recursively handle breakpoints, i.e. set breakpoints and evaluate an expression while perhaps even at another breakpoint
- Ability to redefine debugging commands, e.g. so I can type `c` and have the debugger print the value of `c` rather than continuing the program

Incorrect Precedence

- Shift operations (<< and >>) and bitwise and (&) should have the same precedence as multiplication and division (* / // %)
- Bitwise or (|) and xor (^) should have the same precedence as addition and subtraction (+ and -)
- Just because C got it wrong doesn't mean Python had to

Constants aren't

- `True` and `False` can be redefined
- `inf` and `nan` aren't reserved words and aren't recognized even though they are the strings printed for `float('inf')` and `float('nan')`, respectively
- `j` is used in complex numbers, but `i` is not so recognized; MATLAB sensibly allows either to be used; as a mathematician, I feel discriminated against.

Python and javascript almost match

Python	Javascript
`True, False`	`true, false`
`None`	`null`
`string.join(list)`	`array.join(string)`
`list[3:7]`	`array.slice(3,7)`
`list[-1]`	`array.slice(-1)[0]`
`for x in list` (x is the element)	`for x in array` (x is the index)
`{} []` are `False`	`{} []` are `true`

datetime.datetime and datetime.date are incomparable

- `datetime.datetime(2000,1,1) != datetime.date(2000,1,1)`
  `True`
- `datetime.datetime(2000,1,1)-datetime.date(2000,1,1)`
  `TypeError: unsupported operand type(s) for -`
- `datetime.datetime(2000,1,2)-datetime.datetime(2000,1,1)`
  `== datetime.date(2000,1,2)-datetime.date(2000,1,1)`
  `True`