BIG DADDY'S PYTHON TOOLBOX For 3.5

R4

© 2016 John A. Oakey
V060516B

Escape Characters

Non-printable characters represented with backslash notation:
\a Bell or alert, **\b** Backspace, **\cx** Control - x, **\C-x** Control-x, **\e** Escape, **\f** Formfeed, **\M-\C-x** Meta-Control-x, **\n** Newline, **\s** Space, **\t** Tab, **\v** Vertical tab, **\x** Character x, **\r** Carriage return, **\nnn** Octal notation, where range of n is 0-7 **\xnn** Hexadecimal notation, n is in the range 0.9, a.f, or A.F

String Format Operator: %

% is used with print to build formatted strings
print ("My horse %s has starting slot %d!" % ('Arrow', 5))
Where the % character can format as:
%c character, **%s** string, **%i** signed decimal integer, **%d** signed decimal integer, **%u** unsigned decimal integer, **%e** exponential notation, **%E** exponential notation, **%f** floating point real number, **%g** the shorter of %f and %e, **%G** the shorter of %f and %E
also: * specifies width, **-** left justification, **+** show sign, **0** pad from left with zero, **(& more)**

Data Containers Methods / Operations

Tuples fixed, immutable sets of data that can not be changed mytup=(7,'yes',6,'no') a 1 element tuple requires a comma xtup=('test',)
Indexing and slicing the same as for stings.
tuple(sequence or list) - converts list to tuples: newtup = tuple(mylist) ;
len(tuple); **max(tuple)**; **min (tuple)**

Dict (dictionary) - a series of paired values.
d = { 'a':'animal', 2:'house', 'car':'Ford', 'num': 68}
d.keys() - value of d; **d.values()**;
d.items() - pairs list; **len(d)**;
d[key] = value; **del d[key]**; **d.clear()** remove all; **key in d**; **key not in d**; **keys()**; **d.copy()** makes a shallow; **fromkeys(seq[, value])** from keys() is a class method - returns a new dictionary value defaults to None.
get(key[, default]) ; **items()**
~~**iteritems()** ; **itervalues()** ; **iterkeys()**~~ ↓
d.items(); **d.values()**; **d.keys()** ←
pop(key[, default]) remove and re-turn its value or default; **popitem()**;
setdefault(key[, default])
update([other])
To find a key if you know the value:
mykey=[key **for** key, value **in** mydict.items()**if** value==theval][0]

Lists
lst[i] = x item lst of s is replaced by x
lst[i:j] = t slice of s from i to j is replaced by the contents of iterable t
del lst[i:j] same as lst[i:j] = []
lst[i:j:k] = t the elements of s[i:j:k] are replaced by those of t
del lst[i:j:k] removes the elements of s[i:j:k] from the list
lst.append(x) appends x to the end of the sequence (same as lst[len(lst):len(lst)] = [x])
lst.clear() removes all items from s (same as del lst[:])
lst.copy() creates a shallow copy of s (same as lst[:])
lst.extend(t) or **s += t** extends lst with the contents of t (for the most part the same as **s**[len(s):len(s)] =t)
lst *= n updates lst with its contents repeated n times
lst.insert(i, x) inserts x into s at the index given by i (same as **lst[i:i] = [x]**)
lst.pop([i]) retrieves the item at i and also removes it from s
lst.remove(x) remove the first item from lst where **lst[i] == x**
lst.reverse() reverses the items of s in place
lst.sort() sort ascending, return None

Arrays - none, use **numpy** or **array** module or forget it.

Sets an unordered collection of unique immutable objects - **no multiple occurrences of the same element**
myset = set("Bannanas are nice"); print(myset)
↳: {'i', 'e', 's', 'a', 'B', ' ', 'c', 'r', 'n'}
add(), **clear()**, **pop()**, **discard()**, **copy difference()**, **remove()**, **isdisjoint()**, **issubset()**, **issuperset()**, **intersection()**
Example: Myset.add('x')

Useful Modules

Good 3rd Party Index:
https://pymotw.com/2/py-modindex.html
Python Standard Library Module Index with links:
https://docs.python.org/2/library/
pip is normally installed with Python but if skipped the **ensurepip** PACKAGE will bootstrap the installer into an existing installation.
python -m pip install SomePackage - command line
sys stdin standard input, stdout std output, exit("some error message")
os deep operating system access .open(name [,mode[, buffering]]) modes: '**r**' reading, '**w**' writing, '**a**' appending, binary append '**b**' like 'rb'
time .asctime(t) .clock() .sleep(secs)
datetime date.today() datetime.now()
random .seed([x]) .choice(seq) .randint (a,b) .randrange(start, stop [, step]) .random() - floating point [0.0 to 1.0]
csv import/export of comma separated values .reader .writer .excel
itertools advanced iteration functions
math like Excel math functions .ceil(x), .fsum (iterable), .factorial(x), .log(x[,base]), pi, e
See also **cmath** for complex numbers
urllib for opening URLs, redirects, cookies, etc
pygame see http://www.pygame.org/hifi.html
tkInter Python's defacto std GUI - look it up
calendar—a world of date options

```
>>> import calendar
>>> c = calendar.TextCalendar(calendar.SUNDAY)
>>> c.prmonth(2016, 9)
   September 2016
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

This only works with a mono-spaced font like Consolas
☜ **output**

curses - does not work in windows
picamera - Python access to your Raspberry Pi camera
RPi.GPIO - control Pi pins via Python
xml - to work with xml files UNSECURE
array or **numpy** work with arrays
Arrayname = array(typecode, [Initializers]
a = numpy.array([[1,2,3,4],[5,6,7,8]])
tarfile / **zipfile** - file compression

multiprocessing - take the course if you can handle it
wave - interface to wav format
yahoo-finance—to get stock data From PyPi **$ pip install yahoo-finance** use for historic data
googlefinance 0.7—real-time stock data **$ pip install googlefinance**

(re)Regular Expresions module

Searching for pattern matches: Top level functions (match, search,etc.) mirror arguments in corresponding compiled pattern method.
Compile ex: **re.compile(pattern)**
mypat=re.**compile**(r'\d..\w') then
myso=mypat.**search**(str) myso is search obj
Search object attributes:
group(), **start()**, **end()**, **span()**
Topline functions ex: myso=re.search (r'\d..\w', str)
search(pat,str)↳ **True or None**
match(pat,str) start of str ↳ **True or None**

```
match = re.search(pattern, string)
if match:
    process(match)
```

fullmatch, **findall**, **escape, purge**
Flags: S (DOTALL), A, I (IGNORECASE), M(MULTILINE ^$), X (VERBOSE), U,
Matching Characters: **use r' to match literally**; in v3 match is Unicode by default
\d any decimal digit \D non-decimal
\w any alphanumeric \W non-alphanum
\s any white space chr \S non-whtspace
any except newline * 0 or more + 1 or more ? 0 or 1 X{n} exactly n ,'X' chars X{m,n} between m & n X's **$** end of str
[ ] contains a set of chars to match
  '-' – a range – [a-c] matches a,b,or c
  special chars lose meaning inside [ ]
  ^ as 1st char starts complimentary match
| OR: a|b matches a OR b (...) whatever re is in the parens (?abcdef) one or more letters in parens (?=...) a look ahead assertion, "only if" (?!=...) negated look-ahead assertion, "not if"
\A match only at start of string
\Z match only end of string \b empty string at the start/end of a word
Modifying Strings:
split() str into a list at re match
sub(pat,repl,str) - repl can be a function
subn() like sub but ↳ the new str

Operators

Math: **+**, **-**, *, **/**, **//** (floor or truncated division), ** (exponent), **%** (mod or modulo returns the remainder) x = 8%3; print(x) ↳2
Assignment: (execute & assign) =, +=, -=, *=, /=, **=, %=
Boolean/Logical: **and, or, not**
Comparison: <, <=, >, >=, **is**, **is not**, **==** (equal), **!=**(not equal)
Special String: **+** concatenation (repetition), **[]** (slice), **[:]** (range slice), **in** (true if found, if "c" in "cat"), **not in**, **r** (r'*str*' – raw string suppresses ESC characters)
Identity: **is/is not** checks if variables point to the same object
Bitwise: **&**, **|** (or), **^** (xor), **~** (flips), **<<** (shift lft), **>>**(shift rt)
New Soon: **@** - a matrix multiplier
Note: operator module adds more.

comments and suggestions appreciated:
john@johnoakey.com

Basic Programming Examples: http://www.java2s.com/Tutorial/Python/CatalogPython.htm
or https://wiki.python.org/moin/BeginnersGuide/programmers/SimpleExamples