Title:	strpaint Overview	Created:	2022-11-16
Status:	Finished	Author:	Lion Kimbro

Intent

Documentation for strpaint, 2-D string painter.

Progress

2022-11-16	started document; wrote up intent
2022-12-01	reformatted this doc to match updated template layout, divided out API sections, documented matching system

Requirements

- strpaint.py & strscan.py installed

Manifest

file	*requires*	*significance*
`strpaint.py`	strscan.py	2-D painting
`strscan.py`		read in template from 2-D string; typically you won't even know you're using it

Vocabulary

word	*meaning*
clipping region	a region (x,y,x1,y1) that characters can be poked into, inclusive of x1,y1 col/row
space	the global list of text lines; each line is a minimal width

Example of Use

Basic use:

```
>>> import strpaint
>>> strpaint.poke(5,5, "X")
>>> strpaint.show()
    0.       10.       20.       30.       40.       50.
     0123456789    5    0123456789    5    0123456789    5
    \O....o....O....o....O....o....O....o....O....o....O....o....
  0 O
  1 .
  2 .
  3 .
  4 .
  5 O     X
>>> print(strpaint.as_str())





     X
>>>
```

Showing a template:

```
from strpaint import *
import strscan
use_template(strscan.test)
copy_template()
template_write("baz", "this is a test")
```

Here's an example strscan template:

```
“””
  ..foo...............................  This is a literal string
  ....................................
  ....................................  ...bar................................
  ....................................  ......................................
  ....................................  ......................................

  __baz__________
  __boz__________
“””
```

The text "__baz__________" gets replaced with "this is a test". Any text that begins with a single "." or "_" is recognized as a replacement space.

Underscores define individual lines, whereas periods define multi-line regions (2-dimensional regions).

Note that the regions clip, and that text does not WRAP.

API Reference (strpaint)

positioning

fn & args	*description*	*note*
loc()	return X&Y position	
loc(x, y)	set X&Y position	
inspace(x,y)	is the x,y coordinate in existing space?	
xspace(x,y)	expand the space to include this x,y	
dims()	returns dimensions of the space (w,h)	width is *highest* width, but lines in the space *can* have less than this width

clipping

clip()	return clipping rectangle (x,y,x1,y1)	
clip(x,y,x1,y1)	set clipping region	clipping region does… what?
clip0()	set clipping to (0,0) - infinity	

read/write

peek(x,y)	return character in space, or None	if it's outside the clipping region, it returns None
poke(x,y, ch)	poke a character into the space	if it's outside the clipping region, it won't poke in
readrow(x,x1,y, default="")	read a specific row[x:x1] out	if it reads outside the clipping region, it adds the default
readcol(x,y,y1, default="")	read a specific col[y:y1] out	if it reads outside the clipping region, it adds the default
writerow(x,x1,y, s)	write into the specific space	s must be at least as long as x1-x, but if longer, it stops early
writecol(x,y,y1, s)	write into the specific space	s must be at least as long as y1-y, but if longer, it stops early
write(s, flags="f")	write a string at the cursor position	o "stay at original position" e "stay at end" h "home at end" f "following line beginning at end"
clear()	clear the space	
reset()	reset everything, clear everything	

present

chart(flags="")	return a chart with rulers at edges	flag "e" – show ("|") at the horizontal boundary of each line
show()	print(chart())	
as_str()	return the space as a string	

draw shapes

hline(x,x1,y,ch)	draw a horizontal line	x1 is exclusive
vline(x,y,y1,ch)	draw a vertical line	y1 is exclusive
box(x,y,x1,y1,ch)	draw a box	x1 & y1 are exclusive
fill(x,y,x1,y1,ch)	fill a box	x1 & y1 are exclusive
cut(x,y,x1,y1)	return a cut string	x1,y1 exclusive -- cut space is replaced with a space
copy(x,y,x1,y1)	return a copy string	x1,y1 exclusive
paste(x,y, s)	paste a cut or copied string	does not affect cursor position

templates

use_template(template)	use a template string	the template string is fed into strscan
copy_template()	replace the space with the template	
template_clip(label)	clip to a named region in the template	
template_write(label, text)	write to a named region in the template	write is clipped; clipping region & position saved & restored before/after

Template replacement strings begin with at least one "." or "_".
"_" delineates single-line regions, "." delineates multi-line regions.
There is no line-wrapping within regions.
g[PARSED_TEMPLATE] contains the details of scanned regions, in the form: `[(x,x1,y,y1, name, '.' or '_' or 'LIT'), …]`