other_functions

March 30, 2024

1 other FullControl functions

these other functions support the design process in a range of ways

they do not give exhaustive functionality, but highlight some useful concepts

for advanced design, you will likely create new custom functions that do things like these

<this document is a jupyter notebook - if they're new to you, check out how they work: link, link, link>

run all cells in this notebook in order (keep pressing shift+enter)

[ ]:
```
import fullcontrol as fc
```

fc.linspace() linspace is a common function included in numpy and other libraries

it creates a list of evenly spaced numbers between defined start and end numbers

it's been created in FullControl to reduce the need to import other packages

`linspace(start, end, number of points)`

[ ]:
```
print(fc.linspace(1.5,2.5,11))
```

travel to a point with fc.travel_to() in addition to turning the extruder on and off directly with an Extruder object, the 'travel_to' function can be used for the specific case of turning extrusion off, moving to a single point, then turning extrusion on

this function returns a list of three steps: [Extruder(on=False), Point, Extruder(on=True)]

since it returns a list, extend() must be used instead of append() when adding the returned steps to an existing list of steps

[ ]:
```
steps_layer_1 = [fc.Point(x=0, y=0, z=0), fc.Point(x=5, y=20), fc.Point(x=10,␣
↪y=0)]
steps_layer_2 = [fc.Point(x=0, y=0, z=0.4), fc.Point(x=5, y=20), fc.Point(x=10,␣
↪y=0)]
steps = steps_layer_1 + fc.travel_to(steps_layer_2[0]) + steps_layer_2
steps.extend(fc.travel_to(fc.Point(z=5)))
fc.transform(steps, 'plot', fc.PlotControls(color_type='print_sequence'))
```

define relative points with fc.relative_point() define a new point relative to a reference point. this only works if all of x y z attributes are defined for the reference point. it is possible to supply a reference list instead of a reference point, in which case the function uses the last point in the list as the reference point

see the design tips tutorial for a way to define absolute points (P) and relative points (R) extremely concisely with *steps.append(**P**(x,y,z))* and *steps.append(**R**(x,y,z))* respectively

[ ]:
```
steps = [fc.Point(x=50, y=50, z=0.2), fc.Point(x=50, y=60, z=0.2)]
steps.append(fc.relative_point(steps[-1], 10, 0, 0))
steps.append(fc.relative_point(steps, 0, -10, 0))
# to be extra concise, assign the the relative_point function to R
for step in steps: print(step)
```

check a design with fc.check() this function can be used to check a design. the current implementation of this function simply summarises the objects in the design and checks that you haven't accidentally added a list of objects as a single entry in the design. the example below shows how this mistake is easy to make if you used append() instead of extend() when adding a list of extra steps to an existing list of steps

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append(fc.Extruder(on=False))
steps.append(fc.Point(x=10))
steps.append(fc.Extruder(on=False))
steps.append(fc.Fan(speed_percent=85))
steps.append(fc.Point(x=20))
print('check 1:')
fc.check(steps)
steps.extend(fc.rectangleXY(fc.Point(x=20, y=0, z=0), x_size=10, y_size=10)) #␣
 ↪fc.rectangleXY() returns a list of five points
print('\ncheck 2 (extended steps with a list):')
fc.check(steps)
steps.append(fc.rectangleXY(fc.Point(x=20, y=0, z=0), x_size=10, y_size=10))
print('\ncheck 3: (appended steps with a list)')
fc.check(steps)
```

flatten a *design* that contains lists of objects with fc.flatten() FullControl ***designs*** must be 1D arrays of FullControl objects. however, it might be conceptually useful to create a ***design*** as a list of actions, where each action may be formed of several steps. if so, you can use fc.flatten to turn a collection of lists into a 1D list

[ ]:
```
steps = []
steps.append(fc.Point(x=0, y=0, z=0))
steps.append([fc.Point(x=10, y=10), fc.ManualGcode(text="G4 P2000 ; pause for 2␣
 ↪seconds")])
steps.append([fc.Point(x=20, y=20), fc.ManualGcode(text="G4 P2000 ; pause for 2␣
 ↪seconds")])
```

```
steps.append([fc.Point(x=30, y=30), fc.ManualGcode(text="G4 P2000 ; pause for 2␣
 ↪seconds")])
print('original steps:')
for step in steps:
    print(repr(step))

print("\noriginal steps 'check':")
fc.check(steps)

flat_steps = fc.flatten(steps)
print('\nflat steps:')
for step in flat_steps:
    print(repr(step))
print("\nflat steps 'check':")
fc.check(flat_steps)
```

find the first point in a *design* with fc.first_point() this function finds the first point object in the design

it can be set to find the first point object of any kind or the first one with all of x y z defined

[ ]:
```
steps = [fc.Fan(speed_percent=75), fc.Point(x=1), fc.Point(y=3), fc.Point(x=1,␣
 ↪y=1, z=1)]
print("first step in the design: " + str(type(steps[0]).__name__))
print("first point in the design (not fully defined): " + str(fc.
 ↪first_point(steps, fully_defined=False)))
print("first point in the design (fully defined): " + str(fc.first_point(steps,␣
 ↪fully_defined=True)))
```

extract points from a *design* with fc.point_only() this function removes all objects from the ***design*** except Point objects

it's useful for creating plots or analyzing the geometry, etc.

it can return fully defined points (all of x y z defined - carried over from previous points if not set by the user) or it can return the points exactly as they were created

[ ]:
```
steps = [fc.Point(x=1, y=1, z=0), fc.Fan(speed_percent=75), fc.Point(y=3)]
print("steps: " + str(steps))
print("points in steps (no tracking): " + str(fc.points_only(steps,␣
 ↪track_xyz=False)))
print("points in steps (tracking): " + str(fc.points_only(steps,␣
 ↪track_xyz=True)))
```

export and import a *design* as a .json file the export_design() exports a ***design*** to a .json file.

the import_design() function loads the design back into python. it must be passed the FullControl module handle (fc in this notebook) so it can convert the .json file into the correct type of FullControl

objects

importing a ***design*** is useful if it is computationally demanding to generate the python list of FullControl objects. by exporting a design, you can import it and resume work without needing to repeat the computations/calculations

[ ]:
```
steps = [fc.Point(x=0, y=0, z=0), fc.Point(x=10), fc.Extruder(on=False), fc.
  ↪Fan(speed_percent=75), fc.Point(x=20)]
fc.export_design(steps, 'my_design')
steps_imported = fc.import_design(fc, 'my_design')
print(fc.transform(steps_imported, 'gcode'))
```