Functions

`activity_days(period)`

returns a list of boolean values per day to indicate whether there are procedures on the day

- Source code in `trialdesign/td.py`

```
def activity_days(period):
        """returns a list of boolean values per day to indicate whether there are procedures on the day"""
        start = period["start"]
        duration = period["duration"]
        if start <0 and start+duration >0:
                duration+=1

        # start and end of period, start and end of trains of procedure days
        out = [start, start+duration-1]
        for x in ["administrations", "procedures"]:
                if x in period.keys():
                        for i in period[x]:
                                if "days" in i.keys():
                                        temp = decode_daylist(i["days"])
                                        out += extract_start_end(temp)

        # all PK days
        if "procedures" in period.keys():
                for i in period["procedures"]:
                        if "times" in i:
                                out += [d for (d, t, r) in normalize_procedure(extract_procedure(period, i["caption"]))]

        if "intervals" in period.keys():
                for i in period["intervals"]:
                        if "start" in i.keys() and "duration" in i.keys():
                                start = i["start"]
                                duration = i["duration"]
                                if start <0 and start+duration>0:
                                        duration += 1
                                temp = list(range(start, start+duration))
                                if 0 in temp:
                                        temp.remove(0)
                                out += extract_start_end(temp)
        out.sort()
        temp = [False] * period['duration']
        for i in list(dict.fromkeys(out)):
                temp[day_index(period, i)] = True
        return(temp)
```

`add_output(old, new)`

add output of render functions

- Source code in `trialdesign/td.py`

```
def add_output(old, new):
        """add output of render functions"""
        return([o+n for o, n in zip(old, new)])
```

`day_index(period, day)`

convert day to index within daylist

- Source code in `trialdesign/td.py`

```
def day_index(period, day):
        """convert day to index within daylist"""
        temp = day - period['start']
        if period['start'] < 0 and day > 0:
                temp -= 1 # correct for absent day 0
        if temp <0 or temp>period["duration"]-1:
                raise IndexError(f'day index {day} out of range ({period["start"]} to {period["start"]+period["duration"]})')
        return(temp)
```

`decode_daylist(daylist)`

convert 'days' field (including day ranges) to list of individual days

Parameters:

daylist: list of period days, either in numerical format (e.g., - , ,), or as strings that may represent single days (e.g., "- ", " ") or day ranges (e.g., " - "). Day ranges can also include multiple segments (e.g., " - , - ", " - , , ")

- Source code in `trialdesign/td.py`

```
def decode_daylist(daylist):
        """convert 'days' field (including day ranges) to list of individual days

        Arguments:
                daylist: list of period days, either in numerical format (e.g., -1, 1, 2), or as strings that may represent si
        """
        days = []
        if not isinstance(daylist, list):
                daylist = [daylist]
        for i in daylist:
                if isinstance(i, int):
                        days.append(i)
                elif isinstance(i, str):
                        pat_element = r'(\d+)(-(\d+))?'
                        pat = f'({pat_element}(, )*)'
                        m = re.findall(pat, i)
                        if m:
                                for mm in m:
                                        if mm[3] == "":
                                                days.append(int(mm[1]))
                                        else:
                                                for i in range(int(mm[1]), int(mm[3])+1):
                                                        days.append(i)
        return(days)
```

extract_footnotes(period, caption)

extract footnotes for procedures by day, if applicable

- Source code in `trialdesign/td.py`

```
def extract_footnotes(period, caption):
        """extract footnotes for procedures by day, if applicable"""
        out = [[False] * period['duration'], [''] * period['duration'], []]
        def temp(proc, out):
                if 'footnotes' in proc.keys():
                        for f in proc["footnotes"]:
                                if not "days" in f.keys():
                                        raise KeyError(f'no "days" in footnote "{f["text"]}"')
                                else:
                                        if not isinstance(f["days"], list):
                                                daylist = [f["days"]]
                                        else:
                                                daylist = f["days"]
                                        for d in decode_daylist(daylist):
                                                i = day_index(period, d)
                                                out[0][i] = True
                                                if out[1][i]:
                                                        out[1][i] += ","
                                                out[1][i] += str(f['symbol'])
                                                out[2].append([f['symbol'], f['text']])
                return(out)
        return(iterate_over_procedures(period, caption, out, temp))
```

extract_procedure(period, caption)

get specified administration/procedure as list of tuples (day, [times], relative) for individual days

- Source code in `trialdesign/td.py`

```
def extract_procedure(period, caption):
        """get specified administration/procedure as list of tuples (day, [times], relative) for individual days"""
        out = []
        def temp(proc, out):
                if 'times' in proc.keys():
                        t = proc['times']
                elif 'freq' in proc.keys() and proc['freq'] == 'rich':
                        t = [0, 0]
                else:
                        t = [0]
                if 'relative' in proc.keys():
                        rel = proc['relative']
                else:
                        rel = 1
                out += [(d, t, rel) for d in decode_daylist(proc['days'])]
                return(out)
        return(iterate_over_procedures(period, caption, out, temp))
```

extract_start_end(daylist)

from day list, extract start and end days of trains of days

- Source code in `trialdesign/td.py`

```
def extract_start_end(daylist):
        """from day list, extract start and end days of trains of days"""
        last_day = 0
        out = []
        if daylist:
                daylist.sort()
                for i in daylist:
                        if i == daylist[0] or i != last_day+1 and not (last_day == -1 and i == 1):
                                out += [last_day, i]
                        last_day = i
                out += [i]
        out = list(dict.fromkeys(out))
        if 0 in out:
                out.remove(0)
        return(out)
```

has_timescale(period, caption)

test if procedure has timescale in the respective period

- Source code in `trialdesign/td.py`

```
def has_timescale(period, caption):
        """test if procedure has timescale in the respective period"""
        out = []
        def temp(proc, out):
                if 'timescale' in proc.keys():
                        if proc['timescale'] == 'show':
                                out.append(True)
                return(out)
        return(True in iterate_over_procedures(period, caption, out, temp))
```

item_names(periods, item_class)

return list of interval/administration/procedure names for trial

- Source code in `trialdesign/td.py`

```
def item_names(periods, item_class):
        """return list of interval/administration/procedure names for trial"""
        out = []
        for p in periods:
                if item_class in p.keys():
                        for proc in p[item_class]:
                                temp = proc['caption']
                                if not temp in out:
                                        out.append(temp)
        return(out)
```

normalize_procedure(procedure)

break down procedure times to subsequent days, if longer than h

- Source code in `trialdesign/td.py`

```
def normalize_procedure(procedure):
        """break down procedure times to subsequent days, if longer than 24 h"""
        out = []
        for (d, t, rel) in procedure:
                dd = 0
                while t:
                        temp = [i for i in t if i<24]
                        if temp:
                                out.append((d+dd, temp, rel))
                        t = [i-24 for i in t if i>=24]
                        dd += 1
        return(out)
```

period_day_centers(period, xoffset, daywidth_function)

return list of x-coordinates for day centers

- Source code in `trialdesign/td.py`

```
def period_day_centers(period, xoffset, daywidth_function):
        """return list of x-coordinates for day centers"""
        return([start + width / 2 for start, width in zip(period_day_starts(period, xoffset, daywidth_function), daywidth_funct
```

period_day_ends(period, xoffset, daywidth_function)

return list of x-coordinates for day ends

- Source code in `trialdesign/td.py`

```
def period_day_ends(period, xoffset, daywidth_function):
        """return list of x-coordinates for day ends"""
        starts = period_day_starts(period, xoffset, daywidth_function)
        widths = daywidth_function(period)
        return([s+w for s, w in zip(starts, widths)])
```

period_day_starts(period, xoffset, daywidth_function)

return list of x-coordinates for day starts

- Source code in `trialdesign/td.py`

```
def period_day_starts(period, xoffset, daywidth_function):
        """return list of x-coordinates for day starts"""
        out=[xoffset]
        acc = xoffset
        for i in daywidth_function(period):
                acc += i
                out.append(acc)
        return out[:-1]
```

render_daygrid(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True)

render svg output for the day grid for a period. Output is [svg_output, height]

- Source code in `trialdesign/td.py`

```
def render_daygrid(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True):
        """render svg output for the day grid for a period. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        y = yoffset

        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, height, metrics)

        for start, width, center, label, shading in zip(period_day_starts(period, xoffset, daywidth_function), daywidth_functic
                if shading:
                        svg_out += svg_rect(start, y, width, height, lwd=0, fill_color="lightgray")
                if width > textwidth_function("XX")/3:
                        svg_out += svg_rect(start, y, width, height, lwd=lwd)
                else:
                        svg_out += svg_line(start, y, start+width, y, lwd=lwd, dashed=True)
                        svg_out += svg_line(start, y+height, start+width, y+height, lwd=lwd, dashed=True)
                label = str(label)
                delta = textwidth_function("1")*.5 if label and label[0] == "1" else 0
                if width>textwidth_function(str(label)):
                        svg_out += svg_text(center - textwidth_function(str(label)) / 2-delta, yoffset + height - (height- text
        return([svg_out, height+ypadding*2])
```

render_dose_graph(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True)

render dose over time for administration. Output is [svg_output, height]

- Source code in `trialdesign/td.py`

```
def render_dose_graph(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True):
        """render dose over time for administration. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, lineheight+ textheight_function("X"), metrics)

        startx = period_day_starts(period, xoffset, daywidth_function)
        endx = period_day_ends(period, xoffset, daywidth_function)
        doses = [i for i in extract_field(period, caption, "dose")]
        doses_num = [i for i in doses if isinstance(i, int) or isinstance(i, float)]
        if len(doses_num):
                maxdose, mindose = max(doses_num), min(doses_num)
                def dosey(dose):
                        return(yoffset + lineheight*0.6 - (dose-mindose)/(maxdose-mindose)*lineheight*0.6)
                # if doses:
                lastx, lasty, lastdose = 0, 0, 0
                lastend = 0
                for (s, e, d) in zip(startx, endx, doses):
                        if type(d)==int or type(d)==float:
                                svg_out += svg_line(s, dosey(d), e, dosey(d), lwd=lwd)
                                if lasty:
                                        svg_out += svg_line(lastx, lasty, s, dosey(d), lwd=lwd)
                                lastx, lasty = e, dosey(d)
                                if d != lastdose:
                                        if lastend + textwidth_function("n") < s:
                                                svg_out += svg_text(s, yoffset + lineheight + textheight_function("X"), str(d))
                                                lastend = s + textwidth_function(str(d))
                                        lastdose = d
        return([svg_out, lineheight+textheight_function("X")+ypadding])
```

render_dummy(period, xoffset, yoffset, lineheight, metrics)

render bounding box for visual debugging purposes. Output is svg code only.

- Source code in `trialdesign/td.py`

```
def render_dummy(period, xoffset, yoffset, lineheight, metrics):
        """render bounding box for visual debugging purposes. Output is svg code only."""
        daywidth_function = metrics[0]
        return(svg_rect(xoffset, yoffset, period_width(period, daywidth_function), lineheight, lwd=0, fill_color="cornsilk"))
```

render_interval(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True)

render interval for procedure. Output is [svg_output, height]

- Source code in `trialdesign/td.py`

```
def render_interval(period, caption, xoffset, yoffset, lineheight, metrics, style, first_pass=True):
        """render interval for procedure. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        y = yoffset + lineheight/2
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, lineheight, metrics)

        if first_pass:
                svg_out += svg_text(5, y + textheight_function(caption) * (1/2 - 0.1), caption)

        # render interval box
        starts = period_day_starts(period, xoffset, daywidth_function)
        ends = period_day_ends(period, xoffset, daywidth_function)
        widths = daywidth_function(period)

        height = 0.4 * lineheight
        if 'intervals' in period.keys():
                for intv in period['intervals']:
                        if intv['caption'] == caption:
                                if "start" in intv.keys() and "duration" in intv.keys():
                                        start_list, duration_list = [intv['start']], [intv['duration']]
                                elif "days" in intv.keys() and isinstance(intv["days"], list):
                                        start_list = decode_daylist(intv["days"])
                                        duration_list = [1 for i in decode_daylist(intv["days"])]
                                else:
                                        raise TypeError(f'{period["caption"]}, interval "{intv["caption"]}"')

                                for start, duration in zip(start_list, duration_list):
                                        startx = starts[day_index(period, start)]
                                        end = start + duration -1

                                        if start <0 and end >0:
                                                end += 1
                                        endx = ends[day_index(period, end)]
                                        if "decoration" in intv.keys():
                                                if intv["decoration"] == "bracketed":
                                                        wo = widths[day_index(period, start)]
                                                        wc = widths[day_index(period, end)]
                                                        svg_out += svg_open_bracket(startx, y, lineheight, wo*.6, xpadding=0, r
                                                        svg_out += svg_close_bracket(endx, y, lineheight, wc*.6, xpadding=0, ra
                                        svg_out += svg_rect(startx, y-height/2, endx-startx, height, lwd=lwd)
        return([svg_out, lineheight+ypadding])
```

render_periodcaption(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True)

render caption for period. The 'caption' input is ignored and the caption field of the input period is used. Output is [svg_output, height]

- Source code in `trialdesign/td.py`

```
def render_periodcaption(period, caption, xoffset, yoffset, height, metrics, style, first_pass=True):
        """render caption for period. The 'caption' input is ignored and the caption field of the input period is used. Output
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, height, metrics)
        xcenter = xoffset + period_width(period, daywidth_function)/2
        svg_out += svg_text(xcenter - textwidth_function(str(period['caption']))/2, yoffset+ height - (height-textheight_functi
        return([svg_out, height+ypadding/2])
```

render_periods(periods, x, y, caption, height, render_function, metrics, style, dashes=False, footnotes=False, **kwargs)

applies rendering function to all periods

- Source code in `trialdesign/td.py`

```
def render_periods(periods, x, y, caption, height, render_function, metrics, style, dashes=False, footnotes=False, **kwargs):
        """applies rendering function to all periods"""
        daywidth_function= metrics[0]
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        w = [period_width(i, daywidth_function) for i in periods]
        first = True
        last = False
        h = 0
        out = ""

        # render labels, if applicable
        has_labels = len([i for ii in [extract_labels(p, caption) for p in periods] for i in ii if i != '']) != 0
        has_footnotes = True in [i for ii in [extract_footnotes(p, caption)[0] for p in periods] for i in ii]
        if not footnotes:
                has_footnotes = False

        if has_labels or has_footnotes:
                xx = x
                for p in periods:
                        [svg_out, y_out] = render_labels_footnotes(p, caption, xx, y, height, metrics, style, footnotes=footnot
                        out += svg_out
                        xx += period_width(p, daywidth_function) + periodspacing
                h += lineheight
                y += h

        # render procedure
        for p in periods:
                if p==periods[-1]:
                        last=True

                [svg_out, y_out] = render_function(p, caption, x, y, height, metrics, style, first_pass=first, **kwargs)
                out += svg_out

                if dashes and not last:
                        out += svg_line(x+period_width(p, daywidth_function), y+height/2, x+period_width(p, daywidth_function)+
                x += period_width(p, daywidth_function) + periodspacing
                first=False
        return(add_output(["", h], [out, y_out]))
```

render_procedure(period, caption, xoffset, yoffset, lineheight, metrics, style, default_symbol='diamond', first_pass=True)

render procedure. Output is [svg_output, height]

- Source code in `trialdesign/td.py`

```
def render_procedure(period, caption, xoffset, yoffset, lineheight, metrics, style, default_symbol="diamond", first_pass=True):
        """render procedure. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        svg_out = ""
        if debug:
                svg_out += render_dummy(period, xoffset, yoffset, lineheight, metrics)

        y = yoffset + lineheight/2 # center of the line
        if first_pass:
                svg_out += svg_text(5, y + textheight_function(caption) * (1/2 - 0.1), caption)

        centers = period_day_centers(period, xoffset, daywidth_function)
        widths = daywidth_function(period)
        brackets = extract_field(period, caption, "decoration")
        symbols = procedure_symbols(period, caption, default_symbol)
        dlabels = day_labels(period)
        values = extract_field(period, caption, "value")

        ellipses = [1 if (s!="" and l == "" and len(symbols)>3) else 0 for (s,l) in zip(symbols, dlabels)]

        for p, w, s, b, e, v in zip(centers, widths, symbols, brackets, ellipses, values):
                if s:
                        if e==1 and b=="" and ellipsis:
                                svg_out += svg_circle(p, y, lineheight/30, fill_color="black")
                        elif v != "":
                                if v == 0:
                                        svg_out += svg_symbol(p, y, w*.5, "circle", fill=False, fill_color="none", lwd=lwd)
                                else:
                                        svg_out += svg_symbol(p, y, w*.5, "circle", fill=True, fill_color="black")
                        else:
                                svg_out += svg_symbol(p, y, w, s, size=textheight_function("X"), lwd=lwd, title=caption)
                                if b=="bracketed":
                                        svg_out += svg_open_bracket(p, y, lineheight, w*.8, xpadding=0, radius=lineheight/8, lw
                                        svg_out += svg_close_bracket(p, y, lineheight, w*.8, xpadding=0, radius=lineheight/8, l
        return([svg_out, lineheight+ypadding])
```

render_times(period, caption, xoffset, yoffset, lineheight, metrics, style, maxwidth=100)

render timescale for procedure. Output is [svg_output, height]

- Source code in `trialdesign/td.py`

```
def render_times(period, caption, xoffset, yoffset, lineheight, metrics, style, maxwidth=100):
        """render timescale for procedure. Output is [svg_output, height]"""
        (daywidth_function, textwidth_function, textheight_function) = metrics
        (periodspacing, lineheight, ypadding, lwd, ellipsis, debug) = style

        out = ""
        proc = normalize_procedure(extract_procedure(period, caption))

        ts_days = []
        for x in ['procedures', 'administrations']:
                if x in period.keys():
                        for p in period[x]:
                                if p['caption'] == caption:
                                        if "timescale" in p.keys() and p["timescale"]=="show":
                                                ts_days.append(p["relative"])
        ts_days = set(ts_days)

        y = yoffset
        bracketheight = lineheight * 2/3

        last_scale_end = 0
        if ts_days:
                ## curly brackets
                if debug:
                        out += render_dummy(period, xoffset, y, bracketheight, metrics)

                for ts_d in ts_days:
                        times = unnormalize_procedure([i for i in proc if i[2]==ts_d])[0][1]
                        startx = period_day_starts(period, xoffset, daywidth_function)[day_index(period, min([i for (i, t, rel)
                        endx = period_day_ends(period, xoffset, daywidth_function)[day_index(period, max([i for (i, t, rel) in
                        radius = bracketheight/2
                        if radius * 4 > endx-startx:
                                startx -= radius/2
                                endx += radius/2
                                radius = (endx-startx)/5
                        out += svg_curly_up(startx, endx, y, radius=radius, lwd=lwd)
                y += bracketheight + ypadding*1.5

                ## timescales
                if debug:
                        out += render_dummy(period, xoffset, y, lineheight*1.33 + ypadding*2 + textheight_function("X"), metric
                for ts_d in ts_days:
                        times = unnormalize_procedure([i for i in proc if i[2]==ts_d])[0][1]

                        maxtime = max(times)
                        break_time = min(sorted(list([i for i in times if i<24]))[-1] + 2, 23)
                        times_below = len([i for i in times if i<=break_time])
                        times_above = len([i for i in times if i>break_time])

                        startx = period_day_starts(period, xoffset, daywidth_function)[day_index(period, min([i for (i, t, rel)

                        ### scale
                        scale_height = lineheight/3
                        scale_width = min(len(times) * textwidth_function("XX"), maxwidth-xoffset)
                        scale_break = scale_width * times_below/(times_below+times_above)
                        scale_gap = textwidth_function("m")

                        scale_startx = max(min(startx, xoffset + period_width(period, daywidth_function) - scale_width), xoffse
                        if scale_startx < last_scale_end:
                                y += lineheight*1.33 + ypadding*3 + textheight_function("X")

                        def render_scale(x, y, width, height, scale_min, scale_max, scale_labels, show_unit=False):
                                out = svg_line(x, y, x+width, y, lwd=lwd)
                                label_widths = [textwidth_function(str(i)) for i in scale_labels]
                                last_label_end = 0
                                final_label_begin = x + width - label_widths[-1]/2
                                min_delta = textwidth_function(".")

                                for i, wi in zip(scale_labels, label_widths):
                                        xi = (i-scale_min) * width/(scale_max-scale_min) + x
                                        out += svg_line(xi, y-height/2, xi, y+height/2, lwd=lwd)
                                        dxi = wi/2
                                        if xi-dxi > last_label_end and xi+dxi < final_label_begin - min_delta:
                                                out += svg_text(xi-dxi, y+height/2+textheight_function("X")+ypadding, str(i))
                                                last_label_end = xi+dxi+min_delta
                                        if i == scale_labels[-1]:
                                                temp = str(i)
                                                if show_unit:
                                                        temp += " h"
```

```
                                        out += svg_text(xi-dxi, y+height/2+textheight_function("X")+ypadding, temp)
                        return(out)

                def render_points(x, y, width, scale_min, scale_max):
                        points = [t for t in times if t>=scale_min and t<=scale_max]
                        points_x = [(i-scale_min) * width/(scale_max-scale_min) + x for i in points]
                        out = ""
                        for p, xi in zip(points, points_x):
                                out += svg_symbol(xi, y + lineheight/2, 0, "diamond", size=textheight_function("X"), lv
                        return(out)

                out += render_points(scale_startx, y, scale_break, 0, break_time)
                out += render_points(scale_startx+scale_break+scale_gap, y, scale_width - scale_gap - scale_break, 24,

                out += render_scale(scale_startx, y+lineheight+ypadding, scale_break, scale_height, 0, break_time, rang
                if maxtime >=24:
                        out += render_scale(scale_startx+scale_break+scale_gap, y+lineheight+ypadding, scale_width - sc
                last_scale_end = scale_startx + scale_width

        return([out, y+lineheight*1.33 + ypadding*3 + textheight_function("X")-yoffset])
```

unnormalize_procedure(procedure)

collate procedure times into single day, if relative to the same day

- Source code in `trialdesign/td.py`

```
def unnormalize_procedure(procedure):
        """ collate procedure times into single day, if relative to the same day"""
        out = []
        if procedure:
                rels = set([r for (d, ts, r) in procedure])
                for rel in rels:
                        current_times = []
                        for (d, ts, r) in procedure:
                                for t in ts:
                                        if r==rel:
                                                current_times.append(t+(d-r)*24)
                        out.append((rel, current_times, rel))
        return(out)
```