# `autofletcher`

This module provides functions to (sort of) abstract away manual placement of coordinates by leveraging typst's partial function application.

## Contents

## 1 Introduction

The main entry-point is `place_nodes`, which returns a list of indices and a list of partially applied `node` functions, with the pre-calculated positions.

### 1.1 About placers

A placer is a function that takes the index of current child, and the total number of children, and returns the coordinates for that child relative to the parent.

There is a helper function `placer` which allows easily creating placers from a list of positions. This should be good enough for most uses. See this example

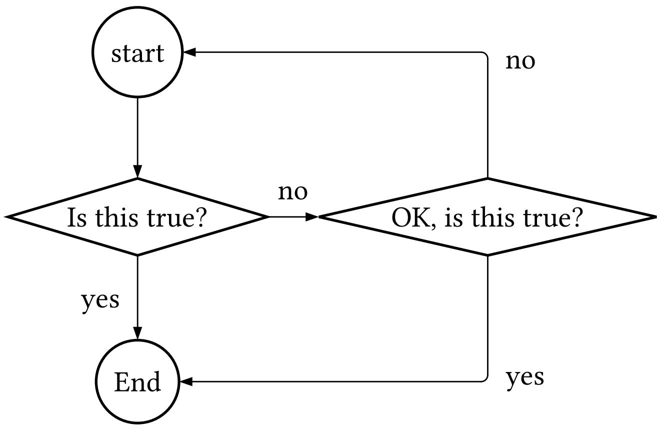
There's also a built-in placer for tree-like structures, `tree_placer`. See this example

It's also relatively easy to create custom placers if needed. See here

# 2 Examples

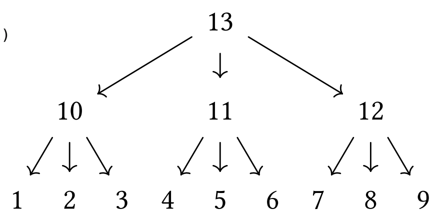
## 2.1 Flowchart

```
#diagram(
  spacing: (0.2cm, 1.5cm),
  node-stroke: 1pt,
  {
    let r = (0, 0)
    let flowchart_placer = placer((0, 1), (1, 0))

    node(r, [start], shape: shapes.circle)
    // question is a node function with the position pre-applied
    let ((iquestion, ), (question, )) = place_nodes(r, 1, flowchart_placer, spread: 20)

    question([Is this true?], shape: shapes.diamond)
    edge(r, iquestion, "-|>")

    let ((iend, ino), (end, no)) = place_nodes(iquestion, 2, flowchart_placer, spread: 10)

    end([End], shape: shapes.circle)
    no([OK, is this true?], shape: shapes.diamond)

    edge(iquestion, iend, "-|>", label: [yes])
    edge(iquestion, ino, "-|>", label: [no])

    edge(ino, iend, "-|>", label: [yes], corner: right)

    edge(ino, r, "-|>", label: [no], corner: left)

  })
```



## 2.2 Tree diagram

```
#diagram(
spacing: (0.0cm, 0.5cm),
{
  let r = (0, 0)
  node(r, [13])

  let (idxs0, (c1, c2, c3)) = place_nodes(r, 3, tree_placer, spread: 10)

  c1([10])
  c2([11])
  c3([12])

  edges(r, idxs0, "->")

  for (i, parent) in idxs0.enumerate() {
    let (idxs, (c1, c2, c3)) = place_nodes(parent, 3, tree_placer, spread: 2)

    c1([#(i * 3 + 1)])
    c2([#(i * 3 + 2)])
    c3([#(i * 3 + 3)])

    edges(parent, idxs, "->")
  }
})
```



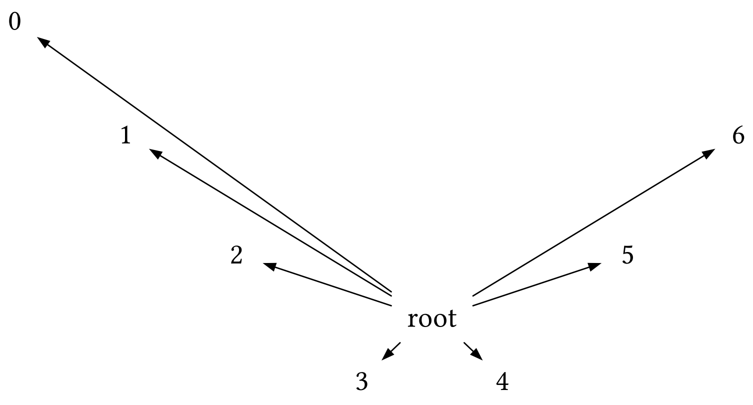
## 2.3 Custom placers

If the built-in placers don't fit your needs, you can create a custom placer; that is, a function that calculates the relative positions for each child. It should accept, in order:

1. (`int`) the index of the child
2. (`int`) the total number of children

and it should return a pair of coordinates, `(x, y)`.

```
#let custom_placer(i, num_total) = {
  // custom logic here
  let x = i - num_total/2
  let y = calc.min(- x, + x) + 1
  return (x, y)
}

#diagram({
  let r = (0, 0)
  node(r, [root])

  let (idxs, nodes) = place_nodes(r, 7, custom_placer, spread: 1)
  for (i, ch) in nodes.enumerate() {
    ch([#i])
    edge(r, idxs.at(i), "-|>")
  }
})
```



# 3 API reference

### edges

Convenience function that draws edges between a parent node and its children, given the coordinates of the parent and children.

#### Parameters

```
edges(
  parent: coordinates,
  children: array of coordinates,
  ..options: any
)
```

**parent** `coordinates`

The coordinates of the parent node

**children** `array of coordinates`

The coordinates of the children nodes

**..options** `any`

Additional options to pass to `edge`

### place_nodes

Calculates the positions of `num_children` children of `parent` node.

Returns a pair of arrays. The first array contains the coordinates of the children, and the second array contains the nodes partially applied with the calculated positions.

#### Parameters

```
place_nodes(
  parent: coordinates,
  num_children: int,
  placer: function,
  spread: int
) -> (array of coordinates + array of nodes)
```

**parent** `coordinates`

The coordinates of the parent node

**num_children** `int`

The number of children to place

**placer** `function`

The function to calculate the relative positions of the children

**spread** `int`

A multiplier for the x coordinate, “spreads” children out. Increase this for high parent nodes.

Default: `1`

### placer

Returns a generic placer, where children are placed according to the given relative positions. If more children are present than there are positions, positions are repeated.

This is probably sufficient for most use cases.

**Parameters**

```
placer(..placements: coordinates ) -> function
```

**..placements** `coordinates`

Relative positions to assign to children

### tree_placer

Calculates the relative position of a child node, like in a tree

Don’t call this directly; instead, pass this as a parameter to `place_nodes`.

**Parameters**

```
tree_placer(
  i: int ,
  num_total: int
)
```

**i** `int`

The index of the child node

**num_total** `int`

The total number of children