Exercise 4.1.1

Let's build our stack diagram by parts.

First, we have defined some variables in `_main_`:

main	radius ⟶ 100 bob ⟶ turtle.Turtle()

I have some doubts at this point. Before the file `polygon.py` runs `circle(bob,radius)`, it applies some `turtle` methods on the variable `bob`. I don't know if this should be present in the stack diagram and, if so, how to do it. The methods are:

```
bob.pu() #pulls the pen up
bob.fd(radius) #'moves' bob radius units forward
bob.lt(90) #'turns' bob 90 degrees lefts
bob.pd() #puts the pen down
```

I don't know if these methods really change `bob`, because when I print `bob` along each step, I always get the same

```
<turtle.Turtle object at 0x06950F10>
```

However, when we look at the Python Turtle screen, we see that the pointer has indeed moved. In order to simplify things, let's just consider `bob` as `bob = turtle.Turtle()`.

main	radius ⟶ 100 bob ⟶ turtle.Turtle()
circle	t ⟶ turle.Turtle() r ⟶ 100
arc	t ⟶ turtle.Turtle() r ⟶ 100 angle ⟶ 360 arc_length ⟶ 628.3185307179587 n ⟶ 160 step_length ⟶ 3.9269908169872414 step_angle ⟶ 2.25

and again the object `t` is modified by a method: `t.lt(step_angle/2)`. Of course, these changes are important for the result. I don't know if this is the standard way of denoting this in a stack diagram, but let's do it in this way:

`_main_`	`radius` ⟶ `100`
	`bob` ⟶ `turtle.Turtle()`
	`bob.pu()`
	`bob.fd(radius)`
	`bob.lt(90)`
	`bob.pd()`

`circle`	`t` ⟶ `bob`
	`r` ⟶ `100`

`arc`	`t` ⟶ `bob`
	`r` ⟶ `100`
	`angle` ⟶ `360`
	`arc_length` ⟶ `628.3185307179587`
	`n` ⟶ `160`
	`step_length` ⟶ `3.9269908169872414`
	`step_angle` ⟶ `2.25`
	`t.lt(1.125)`

`polyline`	`t` ⟶ `t`
	`n` ⟶ `160`
	`length` ⟶ `3.9269908169872414`
	`angle` ⟶ `2.25`
	`for i in range(160)`
	`    t.fd(3.9269908169872414)`
	`    t.lt(2.25)`

	`t.rt(1.125)`

Exercise 4.1.2

Probably the author refers to `t.lt(step_angle/2)` and `t.rt(step_angle/2)`. Without `t.lt(step_angle/2)`, the pointer heads forward, drawing a (though small) horizontal piece of line and when the pointer gets back to the initial point it also comes with approximately horizontal direction. In this way, we have a doubled “horizontal” piece of circle.

Using `t.lt(step_angle/2)` makes the pointer turn a bit to the left in advance, to avoid this horizontal behavior. Then, before the end, `t.rt(step_angle/2)` compensates that turning a bit to the right.