LCG\_turtle

Generated by Doxygen 1.8.14

# **Contents**

1				https://docs.python.org/3/library/turtle.html" $>$ Turtle Graphics $<$ /a $>$ - A set os for practicing CG concepts.	f sim-	1
	1.1	releas	e.notes .			1
2	Nam	nespace	Index			3
	2.1	Name	space List	t		3
3	Clas	s Index	<b>C</b>			5
	3.1	Class	List			5
4	File	Index				7
	4.1	File Lis	st			7
5	Nam	nespace	Docume	entation		9
	5.1	flailDri	ver Names	espace Reference		9
		5.1.1	Function	n Documentation		9
			5.1.1.1	bgcolor()		9
			5.1.1.2	setup()		10
			5.1.1.3	setworldcoordinates()		10
			5.1.1.4	speed()		10
			5.1.1.5	title()		10
			5.1.1.6	window_height()		10
			5.1.1.7	window_width()		11
	5.2	polar N	Namespac	ce Reference		11

ii CONTENTS

5.2.1	Detailed	Description	. 14
5.2.2	Function	Documentation	. 15
	5.2.2.1	_toDenary()	. 15
	5.2.2.2	_toDenaryRec()	. 15
	5.2.2.3	ascension()	. 15
	5.2.2.4	axes()	. 16
	5.2.2.5	BAM2float()	. 16
	5.2.2.6	crossprod()	. 17
	5.2.2.7	crossprod3()	. 17
	5.2.2.8	drawAxis()	. 17
	5.2.2.9	drawBox()	. 18
	5.2.2.10	f0()	. 18
	5.2.2.11	f12()	. 19
	5.2.2.12	f14()	. 20
	5.2.2.13	f15()	. 20
	5.2.2.14	f19()	. 21
	5.2.2.15	f20()	. 22
	5.2.2.16	f21()	. 23
	5.2.2.17	f23()	. 23
	5.2.2.18	f24()	. 24
	5.2.2.19	f25()	. 24
	5.2.2.20	f6()	. 25
	5.2.2.21	float2BAM()	. 25
	5.2.2.22	float2UBAM()	. 26
	5.2.2.23	help()	. 28
	5.2.2.24	initPointList()	. 28
	5.2.2.25	main()	. 29
	5.2.2.26	move()	. 29

CONTENTS

	5.2.2.27	polar2Cartesian()	 30
	5.2.2.28	polarRose()	 30
	5.2.2.29	toBinary()	 32
	5.2.2.30	toDenary()	 32
	5.2.2.31	toFloat()	 33
	5.2.2.32	toInt()	 33
	5.2.2.33	toUBAM()	 33
	5.2.2.34	updateBBOX()	 34
5.2.3	Variable I	Documentation	 34
	5.2.3.1	toDebug	 35
	5.2.3.2	bam_bit_table	 35
	5.2.3.3	BSCALE	 35
	5.2.3.4	cartesian2Polar	 35
	5.2.3.5	clamp	 36
	5.2.3.6	curveList	 36
	5.2.3.7	dotprod	 36
	5.2.3.8	$f1\ldots\ldots\ldots\ldots\ldots\ldots\ldots$	 37
	5.2.3.9	f10	 37
	5.2.3.10	f11	 38
	5.2.3.11	f13	 38
	5.2.3.12	f16	 39
	5.2.3.13	f17	 39
	5.2.3.14	f18	 40
	5.2.3.15	f2	 40
	5.2.3.16	f22	 40
	5.2.3.17	f3	 41
	5.2.3.18	f4	 41
	5.2.3.19	f5	 41

iv CONTENTS

	5	.2.3.20	f7	 41
	5	.2.3.21	f8	 42
	5	.2.3.22	f9	 42
	5	.2.3.23	float2Int	 43
	5	.2.3.24	HEIGHT	 43
	5	.2.3.25	int2Float	 43
	5	.2.3.26	LH	 43
	5	.2.3.27	LSB	 44
	5	.2.3.28	LW	 44
	5	.2.3.29	NBITS	 44
	5	.2.3.30	pointList	 44
	5	.2.3.31	setColor	 45
	5	.2.3.32	staircase	 45
	5	.2.3.33	subvec	 45
	5	.2.3.34	usingFlail	 45
	5	.2.3.35	veclen	 46
	5	.2.3.36	WIDTH	 46
	5	.2.3.37	wrapPi	 46
	5	.2.3.38	WSIZE	 46
	5	.2.3.39	Xc	 47
	5	.2.3.40	Yc	 47
5.3	turtle Nar	mespace	Reference	 47
	5.3.1 F	unction [	Documentation	 47
	5	.3.1.1	bgcolor()	 48
	5	.3.1.2	setup()	 48
	5	.3.1.3	setworldcoordinates()	 48
	5	.3.1.4	speed()	 48
	5	.3.1.5	title()	 49
	5	.3.1.6	window_height()	 49
	5	.3.1.7	window_width()	 49

CONTENTS

6	Clas	s Docu	nentation	51
	6.1	flailDriv	er.FlailDriver Class Reference	51
		6.1.1	Detailed Description	52
		6.1.2	Constructor & Destructor Documentation	52
			6.1.2.1init()	52
		6.1.3	Member Function Documentation	52
			6.1.3.1del_()	52
			6.1.3.2 ascend()	52
			6.1.3.3 backward()	53
			6.1.3.4 color()	53
			6.1.3.5 descend()	53
			6.1.3.6 dot()	53
			6.1.3.7 forward()	54
			6.1.3.8 goto()	54
			6.1.3.9 home()	54
			6.1.3.10 left()	54
			6.1.3.11 pendown()	54
			6.1.3.12 pensize()	55
			6.1.3.13 penup()	55
			6.1.3.14 repeat()	55
			6.1.3.15 reset()	55
			6.1.3.16 right()	55
			6.1.3.17 setheading()	56
			6.1.3.18 setposition()	56
			6.1.3.19 write()	56
		6.1.4	Member Data Documentation	56
			6.1.4.1 f	56
			6.1.4.2 formati	57

vi CONTENTS

			6.1.4.3	formatu	57
	6.2	turtle.F	lailDriver (	Class Reference	57
		6.2.1	Detailed	Description	58
		6.2.2	Construc	tor & Destructor Documentation	58
			6.2.2.1	init()	58
		6.2.3	Member	Function Documentation	58
			6.2.3.1	del_()	58
			6.2.3.2	ascend()	59
			6.2.3.3	backward()	59
			6.2.3.4	color()	59
			6.2.3.5	descend()	59
			6.2.3.6	dot()	60
			6.2.3.7	forward()	60
			6.2.3.8	goto()	60
			6.2.3.9	home()	60
			6.2.3.10	left()	60
			6.2.3.11	pendown()	61
			6.2.3.12	pensize()	61
			6.2.3.13	penup()	61
			6.2.3.14	repeat()	61
			6.2.3.15	reset()	61
			6.2.3.16	right()	62
			6.2.3.17	setheading()	62
			6.2.3.18	setposition()	62
			6.2.3.19	write()	62
		6.2.4	Member	Data Documentation	62
			6.2.4.1	$f \ldots \ldots \ldots \ldots$	63
			6.2.4.2	formati	63
			6.2.4.3	formatu	63
7	File	Docume	entation		65
'	7.1			Reference	
	7.1			prence	
	7.2			prence	
	7.0	ιαι tiσ.p	y 1 110 1 1016		J
Inc	dex				71

# **Chapter 1**

Python <A HREF="https://docs.python...
org/3/library/turtle.html">Turtle Graphics</a> - A
set of simple python programs for practicing CG
concepts.

The set is made up of 12 small programs in increasing order of difficulty.

The most challenging script aims at drawing some parametric curves, defined in polar coordinates, using the turtle intrinsic coordinate system.

Only the **forward**, **left** and **right** commands should be used (no setposition). This way, one may imagine controlling an airplane or a drone, as if a remote control were being used.

In 3D, this would resemble the ubiquitous yaw, pitch and roll rotation paradigm.

# 1.1 release.notes

These programs run either in python 2.7 or python 3.6

- 1. README
- 2. refman.pdf

To run any program:

- · python prog\_name.py or
- python3 prog name.py

where the prog name is the name of a file, with a python 2 or 3 source code, such as:

• python turtle/polar.py

2	Python <a href="https://docs.python.org/3/library/turtle.html">Turtle Graphics</a> - A set of simple python programs for practicing CG concepts.
	pymon programo for practioning our conceptor

# **Chapter 2**

# Namespace Index

# 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

flailDrive	97	9
polar		
	Plot polar equations	11
turtle		47

4 Namespace Index

# **Chapter 3**

# **Class Index**

# 3.1 Class List

Here are the classes.	structs.	unions	and interfaces	with brief	descriptions

flailDriver.FlailDriver	 	 																	51
turtle.FlailDriver	 	 																	57

6 Class Index

# **Chapter 4**

# File Index

# 4.1 File List

Here is a list of all files with brief descriptions:

flailDriver	.py		 																 			65
polar.py			 																 			65
turtle.py			 																			69

8 File Index

# **Chapter 5**

# **Namespace Documentation**

# 5.1 flailDriver Namespace Reference

# Classes

class FlailDriver

# **Functions**

- def title (s)
- def setup (width, height)
- def bgcolor (r, g, b)
- def window\_width ()
- def window\_height ()
- def setworldcoordinates (x0, y0, x1, y1)
- def speed (v)

# 5.1.1 Function Documentation

# 5.1.1.1 bgcolor()

```
def flailDriver.bgcolor (
          r,
          g,
          b )
```

Definition at line 10 of file flailDriver.py.

```
5.1.1.2 setup()
```

Definition at line 7 of file flailDriver.py.

# 5.1.1.3 setworldcoordinates()

Definition at line 19 of file flailDriver.py.

## 5.1.1.4 speed()

```
\begin{array}{c} \text{def flailDriver.speed (} \\ v \text{ )} \end{array}
```

Definition at line 22 of file flailDriver.py.

# 5.1.1.5 title()

```
\begin{array}{c} \text{def flailDriver.title (} \\ s \text{ )} \end{array}
```

Definition at line 4 of file flailDriver.py.

# 5.1.1.6 window\_height()

```
def flailDriver.window_height ( )
```

Definition at line 16 of file flailDriver.py.

# 5.1.1.7 window\_width()

```
def flailDriver.window_width ( )
```

Definition at line 13 of file flailDriver.py.

# 5.2 polar Namespace Reference

Plot polar equations.

#### **Functions**

def drawAxis (length, c=0, ticks=10)

Draw and put labels onto an axis, which is aligned with the coordinate system.

• def axes (w, h, xc=0, yc=0)

Plot a pair of perpendicular axes.

• def crossprod (p0, p1, p)

Cross product of vectors from point p0=(x0,y0) to p1=(x1,y1) and from p1=(x1,y1) to p0=(x,y).

def crossprod3 (p0, p1, p)

3D cross product of vectors from point p0=(x0,y0,z0) to p1=(x1,y1,z1) and from p1=(x1,y1,z1) to p0=(x,y,z).

• def polar2Cartesian (r, a)

Get cartesian coordinates from polar coordinates.

def updateBBOX (p, BBOX=None)

Update a curve bounding box, by adding a new point to it.

def tolnt (x)

Convert a float number, representing a length, to an integer, with a fixed number of bits.

def toFloat (b)

Get the float representation of an integer number.

• def toBinary (num, count=0, sign="")

Get the binary representation of a number, recursively.

def toDenary (b)

Converts a string representing a binary number to denary.

def \_toDenary (a)

Return the decimal number represented by binary digits in a list.

def \_toDenaryRec (a)

Return the decimal number represented by binary digits in a list.

def toUBAM (x)

Convert an angle to UBAM.

• def float2UBAM (num)

Convert a float number to an Unsigned Binary Angle Measurement (UBAM).

def float2BAM (\_num)

Convert a float number to a signed Binary Angle Measurement (BAM).

def BAM2float (b)

Convert BAM to float.

def f0 (c, t, a=0, b=1/(2 \*pi))

```
Archimedean spiral.
• def f6 (a, t)
      Lemniscate of Bernoulli.
• def f12 (c, t, a=2, b=3)
      Limaçon of Pascal.
• def f14 (a, t)
      Cochleoid.

    def f15 (a, t)

      Fermat's Spiral.
• def f19 (a, t, e=1, l=1)
      Parabola.
• def f20 (a, t)
      Hyperbola.
• def f21 (a, t)
      Ellipse.
• def f23 (a, t)
      Star.
• def f24 (a, t)
      Cannabis.
• def f25 (i, j=None)
      Draw a curve defined by a set of points.

    def initPointList (fname)

      Initialize the PointList.

    def help (j=None)

      Print curve identifications.
• def move (x, y, mode=True)
      Move the turtle to a given point.

    def drawBox (b)

      Draw a box.
• def polarRose (func, turns, initialAng=0.0, title=None, nseg=None)
      Draw some polar equations.

    def ascension (p0, p1)
```

# **Variables**

• bool usingFlail = True

def main (argv=None)
 Main program.

Control the turtle's ascension.

Whether using the flail driver for writing a file, instead of turtle for drawing on screen.

• int WIDTH = 800

Canvas width.

• int HEIGHT = 800

Canvas height.

• int LW = 680

World Coordinate width.

```
• int LH = 680
      World Coordinate height.
• int Xc = 0
      World Coordinate center.
• int Yc = 0
      World Coordinate center.
• bool staircase = True
      Should diagonal lines be treated as a slope?
list pointList = []
      Trajectory points - will only be initialized in main if usingPointList is set to true.
veclen = lambda p,q: sqrt(sum([(u-v)**2 for u,v in zip(p,q)]))
      Length of vector from point p = (x1,y1) to q = (x,y).

    subvec = lambda p0,p1: [(i-j) for i,j in zip(p1,p0)]

      Return vector p1-p0.

    dotprod = lambda p0,p1,p: sum([u*v for u,v in zip(subvec(p0,p1),subvec(p1,p))])

      Dot product of vectors from point p0=(x0,y0) to p1=(x1,y1) and from p1=(x1,y1) to p=(x,y).
clamp = lambda a: min(max(-1,a),1)
      Clamp a value to range [-1,1].

    cartesian2Polar = lambda x,y: (sqrt(x*x+y*y), atan2(-y,-x)+pi)

      Get the polar coordinates from cartesian coordinates.

    setColor = lambda v: joe.color("red" if v < 0 else "blue")</li>

      Set the color of the curve.
• int wrapPi = lambda x: x - 360 * ((x + 180) // 360)
      Wraps an angle to \pm 180.
bool <u>__toDebug</u> = False
      Toggle debugging mode.
• int NBITS = 8
      Number of bits after binary point.
• int BSCALE = 2**NBITS
      Scale for fixed point: 256 or 65536, for instance.

    float2Int = lambda f: int(f*BSCALE)

      Maps a float number to integer.
int2Float = lambda b: ldexp(float(b),-NBITS)
      Maps an integer number to float.
• list bam_bit_table = [ 0.0055, 0.0109, 0.0219, 0.0439, 0.088, 0.1757, 0.3515, 0.703, 1.406, 2.8125, 5.625, 11.25,
  22.5, 45.0, 90.0, 180.0 ]
      BAM bit table.
• int WSIZE = len(bam_bit_table)-1
      Word size for BAM.
int LSB = 2**-WSIZE * 180
      Least Significant Bit.
• int f1 = 4: (a * 2*sin(n*t), t)
      A rose of n or 2n petals.
• int f2 = lambda a, t: f1(a,t,5)
      A rose of five petals.
• int f3 = lambda a, t: f1(a,t,3)
      A rose of three petals.
```

```
• int f4 = lambda a, t: f1(a,t,2)
      A rose of four petals.
• f5 = lambda a, t: (a * cos(t/2.0), t)
      Double Loop.
• f7 = lambda a, t: (a * 2 * sin(t), t)
• f8 = lambda a, t: (a * (2*sin(2*t)+1), t)
      Bowtie.
• f9 = lambda a, t: (a * (cos(5*t)**2 + sin(3*t) + 0.3), t)
      Oscar's butterfly.
• f10 = lambda a, t: (a * (sin(t) + sin(5*t/2)**3), t)
      Crassula Dubia.
• f11 = lambda a, t: (a/6 * (9 - 3*sin(t) + 2*sin(3*t) - 3*sin(7*t) + 5*cos(2*t)), t)
      Majestic butterfly.
• f13 = lambda a, t: ((100*a if t <= 0.01 else a/t), t)
      Hyperbolic Spiral.
• f16 = lambda a, t: (a * (sin (2**t) - 1.7), t)
      A Face.
• f17 = lambda \ a, \ t: (a*0.8 * (2 - 2 * sin(t) + sin(t) * (sqrt(abs(cos(t)) / (sin(t) + 1.4)))), \ t)
      Heart.
• f18 = lambda a, t: (a * (1 - cos(t) * sin(3*t)), t)
      Ameba.
• f22 = lambda a, t: (a * (1 + 2 * sin(t/2)), t)
      Freeth's Nephroid.

    list curveList

      List of tuples with: polar equation function, angle range, initial angle, title and number of segments.
```

# 5.2.1 Detailed Description

Plot polar equations.

**Author** 

Paulo and Flavia Roma

Date

01/01/2019

#### See also

```
https://en.wikipedia.org/wiki/Polar_coordinate_system
http://jwilson.coe.uga.edu/emt668/emat6680.2003.fall/shiver/assignment11/polargraphs.
htm
http://wwwp.fc.unesp.br/~mauri/Down/Polares.pdf
http://jwilson.coe.uga.edu/emat6680fa08/kimh/assignment11hjk/assignment11.
https://elepa.files.wordpress.com/2013/11/fifty-famous-curves.pdf
```

## 5.2.2 Function Documentation

## 5.2.2.1 \_toDenary()

Return the decimal number represented by binary digits in a list.

Non recursive version.

Definition at line 264 of file polar.py.

## 5.2.2.2 \_toDenaryRec()

Return the decimal number represented by binary digits in a list.

Recursive version.

Definition at line 273 of file polar.py.

Referenced by toDenary().

#### 5.2.2.3 ascension()

```
def polar.ascension ( p0, p1 )
```

Control the turtle's ascension.

If pointList is set, then a file is being used to describe the trajectory. In this case, decide whether to use the forward-up method or the staircase method. Forward-up method: move forward until the turtle reaches the second point, then ascend to meet it. Staircase method: move forward-upwards incrementally until the second point is reached.

#### **Parameters**

p0	initial point.
p1	end point.

Definition at line 942 of file polar.py.

References toInt(), and veclen.

Referenced by polarRose().

## 5.2.2.4 axes()

```
def polar.axes ( w, h, xc = 0, yc = 0 )
```

Plot a pair of perpendicular axes.

Definition at line 96 of file polar.py.

References drawAxis(), and move().

Referenced by polarRose().

# 5.2.2.5 BAM2float()

Convert BAM to float.

Adding  $180^{\circ}$  to any angle is analogous to taking its two's complement.

Consider the LSB (least significant bit) of an *n-bit* word to be  $2^{-(n-1)}$ , with most significant bit  $(MSB) = 180^{\circ}$ .

- sum(v \* ((b>>i)&1) for i,v in enumerate(bam\_bit\_table)) =
- $(180 b_{15} + 90 b_{14} + 45 b_{13} + 22.5 b_{12} + ... + 0.0055 b_0) =$
- $180 (b_{15} + b_{14} 2^{-1} + b_{13} 2^{-2} + b_{12} 2^{-3} + ... + b_0 2^{-15}) =$
- $180 * 2^{-15}(b_{15} 2^{15} + b_{14} 2^{14} + b_{13} 2^{13} + b_{12} 2^{12} + ... + b_0) =$
- $180*2^{-15}*b$

The range of any angle  $\theta$  represented this way is:

• 
$$0^{\circ} < \theta < 360 - 180 \times 2^{-(n-1)^{\circ}}$$
.

```
b BAM angle.
```

Returns

a float pointing number: b \* LSB \* MSB.

See also

```
https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html
Software Engineering for Image Processing Systems, Page 154
```

Definition at line 397 of file polar.py.

Referenced by main(), toFloat(), and toUBAM().

## 5.2.2.6 crossprod()

```
def polar.crossprod ( p0, p1, p )
```

Cross product of vectors from point p0=(x0,y0) to p1=(x1,y1) and from p1=(x1,y1) to p0=(x,y).

Definition at line 113 of file polar.py.

References subvec.

Referenced by polarRose().

#### 5.2.2.7 crossprod3()

```
def polar.crossprod3 ( p0, p1, p )
```

3D cross product of vectors from point p0=(x0,y0,z0) to p1=(x1,y1,z1) and from p1=(x1,y1,z1) to p0=(x,y,z).

Definition at line 116 of file polar.py.

References subvec.

# 5.2.2.8 drawAxis()

```
def polar.drawAxis ( length, \\ c = 0, \\ ticks = 10 )
```

Draw and put labels onto an axis, which is aligned with the coordinate system.

The origin is at the middle of the axis and there will be the same number of ticks on each side of the axis.

length	axis length.
С	axis center.
ticks	number of divisions.

Definition at line 82 of file polar.py.

Referenced by axes().

# 5.2.2.9 drawBox()

Draw a box.

Definition at line 796 of file polar.py.

References move().

Referenced by polarRose().

# 5.2.2.10 f0()

```
def polar.f0 ( c, t, a = 0, b = 1/(2*pi) )
```

Archimedean spiral.

The spiral becomes tighter for smaller values of "b" and wider for larger values.

```
• r(\theta) = a + b\theta.
```

The Archimedean spiral has two arms, one for  $\theta>0$  and one for  $\theta<0$ .

• The two arms are smoothly connected at the origin.

The negative arm is shown in red on the accompanying graph.

• Taking the mirror image of this arm across the y-axis will yield the other arm.

С	scale factor to be applied.
t	parametric value.
а	origin of the spiral: (a,0).
b	controls the distance between successive turnings.

# Returns

a tuple  $(r, \theta)$  representing a point in polar coordinates.

## See also

```
https://en.wikipedia.org/wiki/Archimedean_spiral
```

Definition at line 418 of file polar.py.

References setColor.

## 5.2.2.11 f12()

Limaçon of Pascal.

- $r(\theta) = a + b \sin(\theta)$ .
- When a < b: limacon with and inner loop.
- When a > b: dimpled limacon.
- When a >= 2b: convex limacon.
- When a = b: cardioid.

Changing from sine to cosine does not affect the shape of the graph, just its orientation.

- · Equations using sine will be symmetric to the vertical axis,
- · while equations using cosine are symmetric to the horizontal axis.
- The sign of b will also affect their orientation.

# See also

```
https://en.wikipedia.org/wiki/Limaçon
```

Definition at line 528 of file polar.py.

5.2.2.12 f14()

def polar.f14 ( 
$$a$$
,  $t$ )

Cochleoid.

- $r(\theta) = a \frac{\sin(\theta)}{\theta}$ .
- r(0) = a.

Negative angles in red.

# **Parameters**

а	scale to be applied on the curve.
t	parametric value.

## Returns

a tuple  $(r, \theta)$  representing a point in polar coordinates.

See also

```
https://en.wikipedia.org/wiki/Cochleoid
```

Definition at line 560 of file polar.py.

References setColor.

5.2.2.13 f15()

def polar.f15 ( 
$$a$$
,  $t$  )

Fermat's Spiral.

• 
$$r^2(\theta) = a^2 \theta$$
.

• 
$$r(\theta) = \pm a\sqrt{\theta}$$
.

Negative angles (red) return  $r = -a\sqrt{\theta}$ .

а	scale to be applied on the curve.
t	parametric value.

#### Returns

a tuple  $(r,\theta)$  representing a point in polar coordinates.

## See also

```
https://en.wikipedia.org/wiki/Fermat%27s_spiral
```

Definition at line 575 of file polar.py.

References setColor.

#### 5.2.2.14 f19()

Parabola.

• 
$$r(\theta) = \frac{l}{1+ e \cos(\theta)}$$
.

#### **Parameters**

а	scale factor to be applied.
t	parametric value.
е	is the eccentricity of the conic section,
1	is the length of the semi-latus rectum (the distance along the y-axis from the pole to the curve).

Using the equation above, the conic section will always have a focus at the pole.

The directrix will be the line  $x = \frac{1}{e}$ 

• 
$$r = \frac{1}{e} sec(\theta)$$
 in polar form.

Different values of *e* will give different kinds of conic sections:

- If e == 0, then the curve is a circle.
- If 0 < e < 1, then the curve is an ellipse.
- If e == 1, then the curve is a parabola.
- If e > 1, then the curve is a hyperbola.

#### See also

```
https://brilliant.org/wiki/polar-curves/
```

Definition at line 622 of file polar.py.

References setColor.

Referenced by f20(), and f21().

## 5.2.2.15 f20()

```
def polar.f20 ( a, t)
```

Hyperbola.

- $r(\theta) = \frac{1}{1+1.5 \cos(\theta)}$ .
- If  $cos(\theta)=-\frac{1}{1.5}\Rightarrow\theta=acos(-\frac{1}{1.5})=\pm2.30052~{\rm rad}=\pm131.81^{\circ}\Rightarrow$

- 
$$\lim_{\theta \to -2.3^-} r(\theta) = \infty$$

- 
$$\lim_{\theta \to -2.3^+} r(\theta) = -\infty$$

- 
$$\lim_{\theta \to 2.3^-} r(\theta) = \infty$$

- 
$$lim_{\theta \to 2.3^+} r(\theta) = -\infty$$

• Throw an exception (raise ValueError) when  $(2.05 \le \theta \le 2.481)$  or  $(-2.481 \le \theta \le -2.05)$ .

# See also

```
https://brilliant.org/wiki/polar-curves/
```

Definition at line 638 of file polar.py.

References f19().

5.2.2.16 f21()

```
\begin{array}{c} \text{def polar.f21 (} \\ & \text{a,} \\ & \text{t )} \end{array}
```

Ellipse.

```
• r(\theta) = \frac{1}{1+0.5 \cos(\theta)}.
```

See also

```
https://brilliant.org/wiki/polar-curves/
```

Definition at line 649 of file polar.py.

References f19().

5.2.2.17 f23()

```
\begin{array}{c} \text{def polar.f23 (} \\ & \textit{a,} \\ & \textit{t )} \end{array}
```

Star.

• 
$$r(\theta) = \sin^2(1.2\theta) + \cos^3(6\theta)$$

See also

```
https://www.originlab.com/index.aspx?go=products/origin/graphing
```

Definition at line 664 of file polar.py.

References setColor.

# 5.2.2.18 f24()

```
\begin{array}{c} \text{def polar.f24 (} \\ & \textit{a,} \\ & \textit{t )} \end{array}
```

Cannabis.

```
• r(\theta) = (1 + 0.9\cos(8\theta))(1 + 0.1\cos(24\theta))(0.9 + 0.1\cos(200\theta))(1 + \sin(\theta))
```

#### See also

```
https://www.wolframalpha.com/input/?i=(1%2B0.9+cos(8+\theta))+(1%2B0.1+cos(24+\theta))+(0.9%2\leftrightarrow B0.1+cos(200+\theta))+(1%2Bsin(\theta))+polar+-pi+to+pi
```

Definition at line 673 of file polar.py.

References setColor.

## 5.2.2.19 f25()

```
\begin{array}{ll} \text{def polar.f25 (} \\ & i, \\ & j = \textit{None )} \end{array}
```

Draw a curve defined by a set of points.

#### **Parameters**

i	a point index.
j	a dummy parameter.

## Returns

ith point on the list, or anything else if j is passed.

Definition at line 683 of file polar.py.

Referenced by polar2Cartesian().

# 5.2.2.20 f6()

```
def polar.f6 ( a, t)
```

Lemniscate of Bernoulli.

- $r^2(\theta) = 2a^2 \cos(2\theta)$ .
- $r(\theta) = \pm a \sqrt{2 \cos(2\theta)}$ .

The length is 2a. Valid angle ranges into [], and invalid into ().

• [-pi/4, pi/4] (pi/4, 3pi/4) [3pi/4, 5pi/4] (5pi/4 - 7pi/4)

Negative angles in red.

## **Parameters**

а	scale to be applied on the curve.
t	parametric value.

# Returns

0 for invalid angles or  $(r,\theta)$  otherwise.

# See also

```
https://en.wikipedia.org/wiki/Lemniscate_of_Bernoulli
```

Definition at line 466 of file polar.py.

References setColor.

# 5.2.2.21 float2BAM()

```
def polar.float2BAM (
```

Convert a float number to a signed Binary Angle Measurement (BAM).

```
_num a float number.
```

#### Returns

a Short Integer [-32768 to 32767] corresponding to an angle [-180 to 180).

#### See also

```
https://github.com/borcun/bams/blob/master/bams.c
http://electronicstechnician.tpub.com/14091/css/Binary-Angular-Measurement-316.↔
htm
```

Definition at line 371 of file polar.py.

References float2UBAM().

Referenced by toInt().

#### 5.2.2.22 float2UBAM()

Convert a float number to an Unsigned Binary Angle Measurement (UBAM).

BAM data words are specifically designed to represent up to  $360^{\circ}$  of angular displacements in binary form, often in steps, or increments of, as small as the LSB value (  $0.0055^{\circ}$  for a 16 bit word).

This 16-bit word '11001000 00000000'  $(b_{15}*2^{15}+...+b_0*2^0)$  can represent a  $281.25^{\circ}$  angle:

- 281.25 = 180 + 90 + 11.25
- float2UBAM(281.25) =  $(1 << 15) + (1 << 14) + (1 << 11) = 2^{15} + 2^{14} + 2^{11} = 51200$

When set to one, the LSB (Least Significant Bit) is equal to  $0.0055^{\circ}$ , while the MSB (Most Significant Bit) is equal to  $180^{\circ}$ .

The MSB value represents half the maximum value that may be transmitted.

When all 16 bits are set:

- an angle greater than  $359.9939^\circ$  is indicated,
- their sum is 359.9939, and corresponds to the maximum quantity that can be transmitted.

When all bits in the BAM data word are clear (with ZEROS), a  $0^{\circ}$  angle is represented.

BAM words are also used to transmit non-angular values, such as range, length or height.

• When non-angular values are being used, the LSB value contains the smallest step, or increment of the quantity being transmitted.

In C language, integers overflow behavior is different regarding the integer signedness.

Two situations arise: (Basics of Integer Overflow)

- · signed integer overflow : undefined behavior
- unsigned integer overflow : safely wraps around (UINT\_MAX + 1 gives 0)

One way, of mimicking this behaviour in Python, is using uint16 from numpy.

num a float number.

## Returns

an Unsigned short integer [0 to 65535] corresponding to an angle [0 to 360).

## See also

```
https://github.com/borcun/bams/blob/master/bams.c
http://electronicstechnician.tpub.com/14091/css/Binary-Angular-Measurement-316.↔
htm
https://loicpefferkorn.net/2013/09/python-force-c-integer-overflow-behavior/
https://docs.scipy.org/doc/numpy-1.13.0/user/basics.types.html
```

Definition at line 354 of file polar.py.

Referenced by float2BAM(), main(), polarRose(), and toUBAM().

#### 5.2.2.23 help()

```
def polar.help (
j = None)
```

Print curve identifications.

Definition at line 718 of file polar.py.

Referenced by main().

# 5.2.2.24 initPointList()

Initialize the PointList.

#### **Parameters**

•	en lui i i
tname	file name with points.

#### Returns

pointList bounding box.

Definition at line 692 of file polar.py.

References updateBBOX().

Referenced by main().

#### 5.2.2.25 main()

```
def polar.main (
          argv = None )
```

Main program.

#### **Parameters**

# argv list of arguments. argv[1] number of segments to draw a curve. argv[2] scale factor to be applied on all curves.

Definition at line 1019 of file polar.py.

References BAM2float(), turtle.bgcolor(), float2UBAM(), help(), initPointList(), polarRose(), turtle.setworldcoordinates(), turtle.speed(), turtle.title(), turtle.window\_height(), and turtle.window\_width().

#### 5.2.2.26 move()

Move the turtle to a given point.

- > direction of movement
- ] turtle head.

#### **Parameters**

X	coordinate.
У	coordinate.
mode	whether to use setposition, or use only forward, left and right.

Definition at line 767 of file polar.py.

Referenced by axes(), drawBox(), and polarRose().

#### 5.2.2.27 polar2Cartesian()

```
\begin{array}{c} \text{def polar.polar2Cartesian (} \\ & r, \\ & a \text{ )} \end{array}
```

Get cartesian coordinates from polar coordinates.

#### **Parameters**

r	radius.
а	angle.

#### Returns

$$p = (x,y)$$

Definition at line 126 of file polar.py.

References f25().

Referenced by polarRose().

#### 5.2.2.28 polarRose()

Draw some polar equations.

We should keep three points: (x0,y0), (x1,y1), (x2,y2).

- $(x0, y0) \rightarrow (x1, y1)$  is the previous segment.
- $(x1, y1) \rightarrow (x2, y2)$  is the current segment.

First, we calculate the angle  $\gamma$  between these two segments, by using the dot product:

• 
$$c = cos(\gamma) = \frac{(x_1 - x_0, y_1 - y_0)}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}} \cdot \frac{(x_2 - x_1, y_2 - y_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} = \frac{(x_1 - x_0) \cdot (x_2 - x_1) + (y_1 - y_0) \cdot (y_2 - y_1)}{\sqrt{((x_1 - x_0)^2 + (y_1 - y_0)^2)((x_2 - x_1)^2 + (y_2 - y_1)^2)}}$$

•  $\gamma = acos(min(max(c, -1), 1)),$ 

so the turtle makes a left or right turn, based on the sign of the cross product of the two segments:

$$Or_2(P_0,P_1,P_2) = \operatorname{sign} \begin{vmatrix} 1 & 1 & 1 \\ x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \end{vmatrix} = x_1y_2 + x_2y_0 + x_0y_1 - x_1y_0 - x_2y_1 - x_0y_2 = x_1y_2 + x_2y_0 + x_1y_0 + x_1y_0 - x_2y_1 - x_0y_2 = x_1y_0 + x_$$

• 
$$x_1(y_2 - y_0 - y_1 + y_1) - x_2(y_1 - y_0) - x_0(y_2 - y_1) =$$

• 
$$x_1(y_2-y_1) + x_1(y_1-y_0) - x_2(y_1-y_0) - x_0(y_2-y_1) =$$

• 
$$(x_1-x_0)(y_2-y_1)-(y_1-y_0)(x_2-x_1) \Rightarrow$$

- 
$$(x_1 - x_0, y_1 - y_0) \times (x_2 - x_1, y_2 - y_1) = (x_1 - x_0) * (y_2 - y_1) - (y_1 - y_0) * (x_2 - x_1) + (y_2 - x_1) + (y_1 - y_0) * (x_2 - x_1) + (y_2 - x_1) + (y_2$$

- $\star < 0 \rightarrow \text{right turn}$
- $\star > 0 \rightarrow \text{left turn}$

Then, the turtle moves forward by a distance equal to the length of the current segment:  $\sqrt{\left(x_2-x_1\right)^2+\left(y_2-y_1\right)^2}$ 

For the first point, we need the angle  $\alpha \in [0, 2\pi]$  between the *x-axis* and the vector  $(x_1 - x_0, y_1 - y_0)$ :

• 
$$\alpha = atan2(y0 - y1, x0 - x1) + \pi$$
  
•  $atan2(y, x) \to \alpha \in [-\pi, \pi]$   
•  $atan2(-y, -x) + \pi \to \alpha \in [0, 2\pi]$ 

#### **Parameters**

func	equation.
turns	polar angle range (extension).
initialAng	initial polar angle.
title	curve name.
nseg	number of segments.

#### See also

https://en.wikipedia.org/wiki/Inverse\_trigonometric\_functionshttps://docs.python.org/3/library/math.htmlhttps://www.mathsisfun.com/algebra/vectors-dot-product.html

Definition at line 851 of file polar.py.

References ascension(), axes(), cartesian2Polar, clamp, crossprod(), drawBox(), float2UBAM(), move(), polar2 Cartesian(), subvec, toInt(), toUBAM(), updateBBOX(), and veclen.

Referenced by main().

#### 5.2.2.29 toBinary()

Get the binary representation of a number, recursively.

#### **Parameters**

num number to be parsed.	
count	number of bits generated so far.
sign	string holding the sign of the number.

#### Returns

string representing the number, with the [amount] and its set of bits:

- e.g. toBinary(15) -> '[4] 1 1 1 1'
- e.g. toBinary(-215) -> '[8] -1 1 0 1 0 1 1 1 '

Definition at line 240 of file polar.py.

Referenced by toInt(), and toUBAM().

#### 5.2.2.30 toDenary()

Converts a string representing a binary number to denary.

Definition at line 252 of file polar.py.

References \_toDenaryRec().

Referenced by toInt(), and toUBAM().

#### 5.2.2.31 toFloat()

Get the float representation of an integer number.

Definition at line 225 of file polar.py.

References BAM2float(), and int2Float.

Referenced by toInt().

#### 5.2.2.32 tolnt()

```
def polar.toInt ( x )
```

Convert a float number, representing a length, to an integer, with a fixed number of bits.

Since we use polar coordinates, angles should be thought of as going from 0 to 360. Using two decimal places, this means going from 0 to 36000.

- Therefore, 16 bits (2 bytes) are enough. For distances, this is also satisfactory, for radii (scales) not too small.
- To keep the integers small, we use BAM whenever possible, so only 2 bytes are enough.
- For radius < 4, BAM does not provide the appropriate precision for distances, and the curve may not even close.

#### See also

```
https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/\topsilon
Binary_scaling.html
https://en.wikipedia.org/wiki/Binary_scaling
https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-fohttps://www.eecs.umich.edu/courses/eecs373/readings/floating-point-to-fixed.\topsilon
pdf
```

Definition at line 202 of file polar.py.

References float2BAM(), float2Int, toBinary(), toDenary(), and toFloat().

Referenced by ascension(), and polarRose().

#### 5.2.2.33 toUBAM()

```
def polar.toUBAM ( x )
```

Convert an angle to UBAM.

The main advantage is using only two bytes, instead of the eight bytes of a float.

If the goal is to send angles through the net or write them on a file, we save 75% of space this way.

#### **Parameters**

```
x an angle as a float.
```

#### Returns

another float angle, after coding x using two bytes.

#### See also

https://blogs.msdn.microsoft.com/shawnhar/2010/01/04/angles-integers-and-modulo-arithme

Definition at line 289 of file polar.py.

References BAM2float(), float2UBAM(), toBinary(), and toDenary().

Referenced by polarRose().

#### 5.2.2.34 updateBBOX()

```
\label{eq:polar_updateBBOX} \begin{array}{l} \text{def polar.updateBBOX (} \\ & p, \\ & \textit{BBOX = None )} \end{array}
```

Update a curve bounding box, by adding a new point to it.

• [xmin, xmax, ymin, ymax, zmin, zmax]

#### **Parameters**

р	a point.
BBOX	bounding box (list of point coordinates).

#### Returns

a new bounding box, or the given bounding box, with p in it.

Definition at line 147 of file polar.py.

Referenced by initPointList(), and polarRose().

#### 5.2.3 Variable Documentation

```
5.2.3.1 __toDebug__
```

```
bool polar.__toDebug__ = False [private]
```

Toggle debugging mode.

Definition at line 170 of file polar.py.

#### 5.2.3.2 bam bit table

```
list polar.bam_bit_table = [ 0.0055, 0.0109, 0.0219, 0.0439, 0.088, 0.1757, 0.3515, 0.703, 1.406, 2.8125, 5.625, 11.25, 22.5, 45.0, 90.0, 180.0 ]
```

BAM bit table.

```
sum(bam_bit_table) = 359.9939
```

Definition at line 302 of file polar.py.

#### 5.2.3.3 BSCALE

```
int polar.BSCALE = 2**NBITS
```

Scale for fixed point: 256 or 65536, for instance.

Definition at line 176 of file polar.py.

#### 5.2.3.4 cartesian2Polar

```
\verb|polar.cartesian2Polar| = \verb|lambda| x,y: (sqrt(x*x+y*y), atan2(-y,-x)+pi)|
```

Get the polar coordinates from cartesian coordinates.

Definition at line 138 of file polar.py.

Referenced by polarRose().

#### 5.2.3.5 clamp

```
polar.clamp = lambda a: min(max(-1,a),1)
```

Clamp a value to range [-1,1].

Definition at line 119 of file polar.py.

Referenced by polarRose().

#### 5.2.3.6 curveList

list polar.curveList

#### Initial value:

```
1 = [(f0, 12*pi, -6*pi, "Archimedean spiral"),
                     (f1,2*pi,0,"A rose of eight petals"),
                     (f2,pi,0,"A rose of four petals"),
(f3,pi,0,"A rose of three petals"),
                      (f4,2*pi,0,"A rose of five petals"),
                     (f5,4*pi,0,"Double Loop"),
                     (f6,6*pi/4,-pi/4,"Lemniscate of Bernoulli"), (f7,pi,0,"Circle"),
                     (f8,2*pi,0,"Bowtie"),
                      (f9,2*pi,0,"Oscar's butterfly"),
1.0
                      (f10,4*pi,0, "Crassula Dubia"),

(f11,2*pi,0, "Majestic butterfly"),

(f12,2*pi,0, "Limaçon"),
11
13
                      (f13,8*pi,0.25,"Hyperbolic Spiral",120),
14
                      (f14,12*pi,-6*pi,"Cochleoid"),
(f15,12*pi,-6*pi,"Fermat's Spiral"),
1.5
16
                      (f16,2*pi,0,"A Face"),

(f17,2*pi,0,"A Heart"),

(f18,2*pi,0,"Ameba"),

(f19,8*pi/5,-4*pi/5,"Parabola"),
17
18
19
20
                       (f20,2*pi,-pi,"Hyperbola"),
(f21,2*pi,-pi,"Ellipse"),
21
22
                       (f22,4*pi,0,"Freeth's Nephroid"),
(f23,10*pi,-5*pi,"Star"),
2.3
24
                        (f24,2*pi,-pi,"Cannabis",600),
2.5
26
                        (f25,2*pi,0,"Point List Based")]
```

List of tuples with: polar equation function, angle range, initial angle, title and number of segments.

Definition at line 731 of file polar.py.

#### 5.2.3.7 dotprod

```
polar.dotprod = lambda p0,p1,p: sum([u*v for u,v in zip(subvec(p0,p1),subvec(p1,p))])
```

Dot product of vectors from point p0=(x0,y0) to p1=(x1,y1) and from p1=(x1,y1) to p=(x,y).

Definition at line 110 of file polar.py.

#### 5.2.3.8 f1

```
int polar.fl = 4: (a * 2*sin(n*t), t)
```

A rose of *n* or *2n* petals.

```
• r(\theta) = a \ 2 \sin(n\theta).
```

If *n* is an integer, the curve will be rose-shaped with

- 2n petals if n is even, and
- *n* petals if *n* is odd.

#### **Parameters**

а	scale to be applied on the curve.
t	parametric value $\theta$ .

#### Returns

a tuple  $(r, \theta)$  representing a point in polar coordinates.

#### See also

```
https://en.wikipedia.org/wiki/Rose_(mathematics)
```

Definition at line 434 of file polar.py.

#### 5.2.3.9 f10

```
polar.f10 = lambda a, t: (a * (sin(t) + sin(5*t/2)**3), t)
```

Crassula Dubia.

• 
$$r(\theta) = \sin(\theta) + \sin^3(5\frac{\theta}{2})$$
.

#### See also

Definition at line 504 of file polar.py.

#### 5.2.3.10 f11

```
polar.f11 = lambda a, t: (a/6 * (9 - 3*sin(t) + 2*sin(3*t) - 3*sin(7*t) + 5*cos(2*t)), t)
```

Majestic butterfly.

• 
$$r(\theta) = 9 - 3\sin(\theta) + 2\sin(3\theta) - 3\sin(7\theta) + 5\cos(2\theta)$$
.

#### See also

https://www.desmos.com/calculator/pgyxrshobg

Definition at line 511 of file polar.py.

#### 5.2.3.11 f13

```
polar.f13 = lambda a, t: ((100*a if t \le 0.01 else a/t), t)
```

Hyperbolic Spiral.

• 
$$r(\theta) = \frac{a}{\theta}$$
.

It begins at an infinite distance from the pole in the center (for  $\theta$  starting from zero,  $r = a/\theta$  starts from infinity),

- and it winds faster and faster around as it approaches the pole;
- the distance from any point to the pole, following the curve, is infinite.

The spiral has an asymptote at y = a:

• for t approaching zero the ordinate approaches a, while the abscissa grows to infinity.

#### **Parameters**

	а	scale to be applied on the curve.
ĺ	t	parametric value.

#### Returns

a tuple  $(r, \theta)$  representing a point in polar coordinates.

See also

```
https://en.wikipedia.org/wiki/Hyperbolic_spiral
```

Definition at line 547 of file polar.py.

5.2.3.12 f16

```
polar.f16 = lambda a, t: (a * (sin (2**t) - 1.7), t)
```

A Face.

```
• r(\theta) = \sin(2^{\theta}) - 1.7.
```

See also

```
\verb|https://www.intmath.com/plane-analytic-geometry/8-curves-polar-coordinates.| \leftarrow php|
```

Definition at line 585 of file polar.py.

5.2.3.13 f17

```
polar.f17 = lambda a, t: (a*0.8 * (2 - 2 * \sin(t) + \sin(t) * (sqrt(abs(cos(t)) / (sin(t) + 1. \leftrightarrow 4)))), t)
```

Heart.

• 
$$r(\theta) = 2 - 2\sin(\theta) + \sin(\theta) \frac{\sqrt{|\cos(\theta)|}}{\sin(\theta) + 1.4}$$
.

See also

```
http://mathworld.wolfram.com/HeartCurve.html
```

Definition at line 592 of file polar.py.

5.2.3.14 f18

```
polar.f18 = lambda a, t: (a * (1 - cos(t) * sin(3*t)), t)
```

Ameba.

```
• r(\theta) = 1 - \cos(\theta) \sin(3\theta).
```

See also

```
https://brilliant.org/wiki/polar-curves/
```

Definition at line 599 of file polar.py.

5.2.3.15 f2

```
int polar.f2 = lambda a, t: f1(a,t,5)
```

A rose of five petals.

Definition at line 437 of file polar.py.

5.2.3.16 f22

```
polar.f22 = lambda a, t: (a * (1 + 2 * sin(t/2)), t)
```

Freeth's Nephroid.

```
• r(\theta) = a(1 + 2\sin(\theta/2))
```

See also

```
https://elepa.files.wordpress.com/2013/11/fifty-famous-curves.pdf
```

Definition at line 657 of file polar.py.

#### 5.2.3.17 f3

```
int polar.f3 = lambda a, t: f1(a,t,3)
```

A rose of three petals.

Definition at line 440 of file polar.py.

#### 5.2.3.18 f4

```
int polar.f4 = lambda a, t: f1(a,t,2)
```

A rose of four petals.

Definition at line 443 of file polar.py.

#### 5.2.3.19 f5

```
polar.f5 = lambda a, t: (a * cos(t/2.0), t)
```

Double Loop.

• 
$$r(\theta) = a \cos(\frac{\theta}{2})$$
.

#### See also

 $\verb|http://jwilson.coe.uga.edu/emt668/emat6680.2003.fall/shiver/assignment11/polargraphs. \leftarrow |htm| | |htm| | |htm| | |htm| | |htm| |h$ 

Definition at line 450 of file polar.py.

#### 5.2.3.20 f7

```
polar.f7 = lambda a, t: (a * 2 * sin(t), t)
```

Circle.

Centered at (0,a) and diameter 2a.

• 
$$r(\theta) = 2a \sin(\theta)$$
.

#### **Parameters**

а	radius.
t	parametric value.

#### Returns

a tuple  $(r, \theta)$  representing a point in polar coordinates.

Definition at line 483 of file polar.py.

#### 5.2.3.21 f8

```
polar.f8 = lambda a, t: (a * (2*sin(2*t)+1), t)
```

#### Bowtie.

• 
$$r(\theta) = 2 \sin(2\theta) + 1$$
.

#### See also

 $\verb|http://jwilson.coe.uga.edu/emat6680fa08/kimh/assignment11hjk/assignment11. \leftarrow |html|$ 

Definition at line 490 of file polar.py.

#### 5.2.3.22 f9

```
polar.f9 = lambda a, t: (a * (cos(5*t)**2 + sin(3*t) + 0.3), t)
```

Oscar's butterfly.

• 
$$r(\theta) = \cos^2(5\theta) + \sin(3\theta) + 0.3$$
.

#### See also

Definition at line 497 of file polar.py.

#### 5.2.3.23 float2Int

```
polar.float2Int = lambda f: int(f*BSCALE)
```

Maps a float number to integer.

#### See also

https://www.youtube.com/watch?v=wbxSTxhTmrs&fbclid=IwAR2f04\_45mIFGtIczS←OzbB8nxfqb6SX0pkVlxySfnnBf4n6e8KdKRHXkY2Ihttps://www.cl.cam.ac.uk/teaching/1011/FPComp/fpcomp10slides.pdf?fbclid=←IwAR3G7UxIIrOGxOUJq7IMp8rzZMlBUex-9ZRyCyu\_842LxPDTWaWyb4Xvyjc

Definition at line 182 of file polar.py.

Referenced by toInt().

#### 5.2.3.24 HEIGHT

```
int polar.HEIGHT = 800
```

Canvas height.

Definition at line 59 of file polar.py.

#### 5.2.3.25 int2Float

```
polar.int2Float = lambda b: ldexp(float(b),-NBITS)
```

Maps an integer number to float.

Reconstruct the number with NBITS bits after the binary point.

Definition at line 186 of file polar.py.

Referenced by toFloat().

#### 5.2.3.26 LH

```
int polar.LH = 680
```

World Coordinate height.

Definition at line 64 of file polar.py.

# 5.2.3.27 LSB

```
int polar.LSB = 2**-WSIZE * 180
```

Least Significant Bit.

```
LSB = bam_bit_table[0] = 2**-15*180 = 0.0054931640625
```

Definition at line 312 of file polar.py.

#### 5.2.3.28 LW

```
int polar.LW = 680
```

World Coordinate width.

Definition at line 62 of file polar.py.

#### 5.2.3.29 NBITS

```
int polar.NBITS = 8
```

Number of bits after binary point.

Definition at line 173 of file polar.py.

### 5.2.3.30 pointList

```
list polar.pointList = []
```

Trajectory points - will only be initialized in main if usingPointList is set to true.

Definition at line 73 of file polar.py.

#### 5.2.3.31 setColor

```
polar.setColor = lambda v: joe.color("red" if v < 0 else "blue")</pre>
```

Set the color of the curve.

Definition at line 162 of file polar.py.

Referenced by f0(), f14(), f15(), f19(), f23(), f24(), and f6().

#### 5.2.3.32 staircase

```
bool polar.staircase = True
```

Should diagonal lines be treated as a slope?

Definition at line 71 of file polar.py.

#### 5.2.3.33 subvec

```
polar.subvec = lambda p0,p1: [(i-j) for i,j in zip(p1,p0)]
```

Return vector p1-p0.

Definition at line 107 of file polar.py.

Referenced by crossprod(), crossprod3(), and polarRose().

#### 5.2.3.34 usingFlail

```
bool polar.usingFlail = True
```

Whether using the flail driver for writing a file, instead of turtle for drawing on screen.

Definition at line 49 of file polar.py.

```
5.2.3.35 veclen
```

```
polar.veclen = lambda p,q: sqrt(sum([(u-v)**2 for u,v in zip(p,q)]))
```

Length of vector from point p = (x1,y1) to q = (x,y).

Definition at line 104 of file polar.py.

Referenced by ascension(), and polarRose().

#### 5.2.3.36 WIDTH

```
int polar.WIDTH = 800
```

Canvas width.

Definition at line 57 of file polar.py.

#### 5.2.3.37 wrapPi

```
int polar.wrapPi = lambda x: x - 360 * ((x + 180) // 360)
```

Wraps an angle to  $\pm 180$ .

See also

```
3D Math Primer for Graphics and Game Development, page 241.
```

Definition at line 167 of file polar.py.

#### 5.2.3.38 WSIZE

```
int polar.WSIZE = len(bam_bit_table)-1
```

Word size for BAM.

Definition at line 306 of file polar.py.

# 5.2.3.39 Xc

```
int polar.Xc = 0
```

World Coordinate center.

Definition at line 66 of file polar.py.

#### 5.2.3.40 Yc

```
int polar.Yc = 0
```

World Coordinate center.

Definition at line 68 of file polar.py.

# 5.3 turtle Namespace Reference

# Classes

class FlailDriver

#### **Functions**

- def title (s)
- def setup (width, height)
- def bgcolor (r, g, b)
- def window\_width ()
- def window\_height ()
- def setworldcoordinates (x0, y0, x1, y1)
- def speed (v)

# 5.3.1 Function Documentation

# 5.3.1.1 bgcolor()

```
def turtle.bgcolor ( r, g, b )
```

Definition at line 10 of file turtle.py.

Referenced by polar.main().

#### 5.3.1.2 setup()

Definition at line 7 of file turtle.py.

#### 5.3.1.3 setworldcoordinates()

```
def turtle.setworldcoordinates ( x0, y0, x1, y1)
```

Definition at line 19 of file turtle.py.

Referenced by polar.main().

#### 5.3.1.4 speed()

```
\operatorname{def} turtle.speed ( v )
```

Definition at line 22 of file turtle.py.

Referenced by polar.main().

```
5.3.1.5 title()
```

```
def turtle.title ( s )
```

Definition at line 4 of file turtle.py.

Referenced by polar.main().

# 5.3.1.6 window\_height()

```
def turtle.window_height ( )
```

Definition at line 16 of file turtle.py.

Referenced by polar.main().

#### 5.3.1.7 window\_width()

```
def turtle.window_width ( )
```

Definition at line 13 of file turtle.py.

Referenced by polar.main().

# **Chapter 6**

# **Class Documentation**

# 6.1 flailDriver.FlailDriver Class Reference

#### **Public Member Functions**

- def \_\_init\_\_ (self, shape, visible)
- def setposition (self, x, y)
- def goto (self, x, y)
- def write (self, t)
- def dot (self)
- def penup (self)
- · def home (self)
- def pendown (self)
- def reset (self)
- def pensize (self, val)
- def color (self, c)
- def setheading (self, h)
- def forward (self, dist)
- · def backward (self, dist)
- def left (self, ang)
- · def right (self, ang)
- def ascend (self, dist)
- def descend (self, dist)
- def repeat (self, n, instructions)

#### **Public Attributes**

• f

#### **Static Public Attributes**

- string formati = "(%d);\n"
- string formatu = "(%u);\n"

52 Class Documentation

#### **Private Member Functions**

```
• def __del_ (self)
```

# 6.1.1 Detailed Description

Definition at line 25 of file flailDriver.py.

#### 6.1.2 Constructor & Destructor Documentation

Definition at line 30 of file flailDriver.py.

#### 6.1.3 Member Function Documentation

Definition at line 99 of file flailDriver.py.

References flailDriver.FlailDriver.f.

#### 6.1.3.2 ascend()

```
\begin{array}{c} \text{def flailDriver.FlailDriver.ascend (} \\ & self, \\ & \textit{dist )} \end{array}
```

Definition at line 83 of file flailDriver.py.

References flailDriver.FlailDriver.f, and flailDriver.FlailDriver.write().

#### 6.1.3.3 backward()

Definition at line 71 of file flailDriver.py.

References flailDriver.FlailDriver.f, and flailDriver.FlailDriver.write().

#### 6.1.3.4 color()

```
def flailDriver.FlailDriver.color ( self, c )
```

Definition at line 61 of file flailDriver.py.

#### 6.1.3.5 descend()

Definition at line 87 of file flailDriver.py.

References flailDriver.FlailDriver.f, and flailDriver.FlailDriver.write().

#### 6.1.3.6 dot()

Definition at line 43 of file flailDriver.py.

54 Class Documentation

# 6.1.3.7 forward()

Definition at line 67 of file flailDriver.py.

#### 6.1.3.8 goto()

```
def flailDriver.FlailDriver.goto ( self, \\ x, \\ y \ )
```

Definition at line 37 of file flailDriver.py.

#### 6.1.3.9 home()

```
\begin{tabular}{ll} \tt def flailDriver.FlailDriver.home ( \\ self ) \end{tabular}
```

Definition at line 49 of file flailDriver.py.

#### 6.1.3.10 left()

```
\begin{array}{c} \text{def flailDriver.FlailDriver.left (} \\ & self, \\ & \textit{ang )} \end{array}
```

Definition at line 75 of file flailDriver.py.

References flailDriver.FlailDriver.f, and flailDriver.FlailDriver.write().

#### 6.1.3.11 pendown()

```
\label{eq:continuous} \mbox{def flailDriver.FlailDriver.pendown (} \\ self \mbox{)}
```

Definition at line 52 of file flailDriver.py.

#### 6.1.3.12 pensize()

```
def flailDriver.FlailDriver.pensize ( self, val )
```

Definition at line 58 of file flailDriver.py.

#### 6.1.3.13 penup()

```
\begin{tabular}{ll} \tt def flail Driver. Flail Driver. penup ( \\ self ) \end{tabular}
```

Definition at line 46 of file flailDriver.py.

#### 6.1.3.14 repeat()

Definition at line 91 of file flailDriver.py.

References flailDriver.FlailDriver.f, and flailDriver.FlailDriver.write().

#### 6.1.3.15 reset()

```
def flailDriver.FlailDriver.reset ( self )
```

Definition at line 55 of file flailDriver.py.

#### 6.1.3.16 right()

```
def flailDriver.FlailDriver.right ( self, ang )
```

Definition at line 79 of file flailDriver.py.

References flailDriver.FlailDriver.f, and flailDriver.FlailDriver.write().

56 Class Documentation

#### 6.1.3.17 setheading()

```
def flailDriver.FlailDriver.setheading ( self, \\ h \ )
```

Definition at line 64 of file flailDriver.py.

#### 6.1.3.18 setposition()

```
def flailDriver.FlailDriver.setposition ( self, \\ x, \\ y \ )
```

Definition at line 34 of file flailDriver.py.

#### 6.1.3.19 write()

Definition at line 40 of file flailDriver.py.

Referenced by flailDriver.FlailDriver.ascend(), flailDriver.FlailDriver.backward(), flailDriver.FlailDriver.descend(), flail→Driver.FlailDriver.left(), flailDriver.FlailDriver.repeat(), and flailDriver.FlailDriver.repeat().

#### 6.1.4 Member Data Documentation

#### 6.1.4.1 f

```
flailDriver.FlailDriver.f
```

Definition at line 31 of file flailDriver.py.

Referenced by flailDriver.FlailDriver.\_\_del\_(), turtle.FlailDriver.\_\_del\_(), turtle.FlailDriver.ascend(), flailDriver.Bailcove

#### 6.1.4.2 formati

```
string flailDriver.FlailDriver.formati = "(%d);\n" [static]
```

Definition at line 27 of file flailDriver.py.

#### 6.1.4.3 formatu

```
string flailDriver.FlailDriver.formatu = "(%u);\n" [static]
```

Definition at line 28 of file flailDriver.py.

The documentation for this class was generated from the following file:

flailDriver.py

# 6.2 turtle.FlailDriver Class Reference

#### **Public Member Functions**

- def init (self, shape, visible)
- def setposition (self, x, y)
- def goto (self, x, y)
- def write (self, t)
- def dot (self)
- def penup (self)
- def home (self)
- def pendown (self)
- def reset (self)
- def pensize (self, val)
- def color (self, c)
- def setheading (self, h)
- def forward (self, dist)
- def backward (self, dist)
- def left (self, ang)
- def right (self, ang)
- def ascend (self, dist)
- def descend (self, dist)
- def repeat (self, n, instructions)

## **Public Attributes**

•

58 Class Documentation

# **Static Public Attributes**

```
• string formati = "(%d);\n"
```

string formatu = "(%u);\n"

# **Private Member Functions**

```
• def __del_ (self)
```

#### 6.2.1 Detailed Description

Definition at line 25 of file turtle.py.

#### 6.2.2 Constructor & Destructor Documentation

Definition at line 30 of file turtle.py.

#### 6.2.3 Member Function Documentation

Definition at line 99 of file turtle.py.

References flailDriver.FlailDriver.f, and turtle.FlailDriver.f.

#### 6.2.3.2 ascend()

```
def turtle.FlailDriver.ascend ( self, dist )
```

Definition at line 83 of file turtle.py.

References flailDriver.FlailDriver.f, turtle.FlailDriver.f, and turtle.FlailDriver.write().

#### 6.2.3.3 backward()

```
\begin{tabular}{ll} \mbox{def turtle.FlailDriver.backward (} \\ \mbox{\it self,} \\ \mbox{\it dist )} \end{tabular}
```

Definition at line 71 of file turtle.py.

References flailDriver.FlailDriver.f, turtle.FlailDriver.f, and turtle.FlailDriver.write().

#### 6.2.3.4 color()

```
def turtle.FlailDriver.color ( self, \\ c \ )
```

Definition at line 61 of file turtle.py.

#### 6.2.3.5 descend()

Definition at line 87 of file turtle.py.

References flailDriver.FlailDriver.f, turtle.FlailDriver.f, and turtle.FlailDriver.write().

60 Class Documentation

# 6.2.3.6 dot()

```
\begin{tabular}{ll} $\operatorname{def}$ turtle. Flail Driver. dot & ( \\ & self \end{tabular} \label{fig:privar}
```

Definition at line 43 of file turtle.py.

#### 6.2.3.7 forward()

```
\begin{tabular}{ll} \mbox{def turtle.FlailDriver.forward (} \\ \mbox{\it self,} \\ \mbox{\it dist )} \end{tabular}
```

Definition at line 67 of file turtle.py.

#### 6.2.3.8 goto()

Definition at line 37 of file turtle.py.

### 6.2.3.9 home()

```
\label{eq:continuous} \mbox{def turtle.FlailDriver.home (} \\ self \mbox{)}
```

Definition at line 49 of file turtle.py.

#### 6.2.3.10 left()

Definition at line 75 of file turtle.py.

References flailDriver.FlailDriver.f, turtle.FlailDriver.f, and turtle.FlailDriver.write().

#### 6.2.3.11 pendown()

Definition at line 52 of file turtle.py.

#### 6.2.3.12 pensize()

Definition at line 58 of file turtle.py.

#### 6.2.3.13 penup()

```
\begin{tabular}{ll} \tt def turtle.FlailDriver.penup & ( \\ & self & ) \end{tabular}
```

Definition at line 46 of file turtle.py.

#### 6.2.3.14 repeat()

Definition at line 91 of file turtle.py.

References flailDriver.FlailDriver.f, turtle.FlailDriver.f, and turtle.FlailDriver.write().

#### 6.2.3.15 reset()

Definition at line 55 of file turtle.py.

62 Class Documentation

#### 6.2.3.16 right()

```
\begin{tabular}{ll} def turtle.FlailDriver.right ( \\ self, \\ ang \end{tabular}
```

Definition at line 79 of file turtle.py.

References flailDriver.FlailDriver.f, turtle.FlailDriver.f, and turtle.FlailDriver.write().

# 6.2.3.17 setheading()

```
def turtle.FlailDriver.setheading ( self, \\ h \ )
```

Definition at line 64 of file turtle.py.

#### 6.2.3.18 setposition()

```
def turtle.FlailDriver.setposition ( self, \\ x, \\ y \ )
```

Definition at line 34 of file turtle.py.

#### 6.2.3.19 write()

Definition at line 40 of file turtle.py.

Referenced by turtle.FlailDriver.ascend(), turtle.FlailDriver.backward(), turtle.FlailDriver.descend(), turtle.FlailDriver.cend(), turtle.FlailDriver.repeat(), and turtle.FlailDriver.right().

#### 6.2.4 Member Data Documentation

#### 6.2.4.1 f

turtle.FlailDriver.f

Definition at line 31 of file turtle.py.

Referenced by turtle.FlailDriver.\_\_del\_(), turtle.FlailDriver.ascend(), turtle.FlailDriver.backward(), turtle.FlailDriver.cdescend(), turtle.FlailDriver.left(), turtle.FlailDriver.repeat(), and turtle.FlailDriver.right().

#### 6.2.4.2 formati

```
string turtle.FlailDriver.formati = "(%d);\n" [static]
```

Definition at line 27 of file turtle.py.

# 6.2.4.3 formatu

```
string turtle.FlailDriver.formatu = "(%u);\n" [static]
```

Definition at line 28 of file turtle.py.

The documentation for this class was generated from the following file:

• turtle.py

64 Class Documentation

# **Chapter 7**

# **File Documentation**

# 7.1 flailDriver.py File Reference

#### **Classes**

· class flailDriver.FlailDriver

# **Namespaces**

• flailDriver

#### **Functions**

- def flailDriver.title (s)
- def flailDriver.setup (width, height)
- def flailDriver.bgcolor (r, g, b)
- def flailDriver.window\_width ()
- def flailDriver.window\_height ()
- def flailDriver.setworldcoordinates (x0, y0, x1, y1)
- def flailDriver.speed (v)

# 7.2 polar.py File Reference

# **Namespaces**

polar

Plot polar equations.

66 File Documentation

#### **Functions**

def polar.drawAxis (length, c=0, ticks=10)

Draw and put labels onto an axis, which is aligned with the coordinate system.

• def polar.axes (w, h, xc=0, yc=0)

Plot a pair of perpendicular axes.

• def polar.crossprod (p0, p1, p)

Cross product of vectors from point p0=(x0,y0) to p1=(x1,y1) and from p1=(x1,y1) to p0=(x,y).

def polar.crossprod3 (p0, p1, p)

3D cross product of vectors from point p0=(x0,y0,z0) to p1=(x1,y1,z1) and from p1=(x1,y1,z1) to p0=(x,y,z).

• def polar.polar2Cartesian (r, a)

Get cartesian coordinates from polar coordinates.

def polar.updateBBOX (p, BBOX=None)

Update a curve bounding box, by adding a new point to it.

def polar.toInt (x)

Convert a float number, representing a length, to an integer, with a fixed number of bits.

def polar.toFloat (b)

Get the float representation of an integer number.

def polar.toBinary (num, count=0, sign="")

Get the binary representation of a number, recursively.

def polar.toDenary (b)

Converts a string representing a binary number to denary.

def polar.\_toDenary (a)

Return the decimal number represented by binary digits in a list.

• def polar.\_toDenaryRec (a)

Return the decimal number represented by binary digits in a list.

def polar.toUBAM (x)

Convert an angle to UBAM.

• def polar.float2UBAM (num)

Convert a float number to an Unsigned Binary Angle Measurement (UBAM).

def polar.float2BAM (\_num)

Convert a float number to a signed Binary Angle Measurement (BAM).

def polar.BAM2float (b)

Convert BAM to float.

• def polar.f0 (c, t, a=0, b=1/(2 \*pi))

Archimedean spiral.

• def polar.f6 (a, t)

Lemniscate of Bernoulli.

• def polar.f12 (c, t, a=2, b=3)

Limaçon of Pascal.

def polar.f14 (a, t)

Cochleoid.

def polar.f15 (a, t)

Fermat's Spiral.

• def polar.f19 (a, t, e=1, l=1)

Parabola.

· def polar.f20 (a, t)

```
Hyperbola.
    • def polar.f21 (a, t)
           Ellipse.

    def polar.f23 (a, t)

           Star.
    • def polar.f24 (a, t)
           Cannabis.

    def polar.f25 (i, j=None)

           Draw a curve defined by a set of points.
    • def polar.initPointList (fname)
           Initialize the PointList.
    • def polar.help (j=None)
           Print curve identifications.
    • def polar.move (x, y, mode=True)
           Move the turtle to a given point.

    def polar.drawBox (b)

           Draw a box.
    • def polar.polarRose (func, turns, initialAng=0.0, title=None, nseg=None)
          Draw some polar equations.
    • def polar.ascension (p0, p1)
           Control the turtle's ascension.

    def polar.main (argv=None)

           Main program.
Variables
    • bool polar.usingFlail = True
           Whether using the flail driver for writing a file, instead of turtle for drawing on screen.
    • int polar.WIDTH = 800
           Canvas width.
    • int polar.HEIGHT = 800
           Canvas height.
    • int polar.LW = 680
           World Coordinate width.
    • int polar.LH = 680
           World Coordinate height.
    • int polar.Xc = 0
           World Coordinate center.
    • int polar. Yc = 0
           World Coordinate center.
    • bool polar.staircase = True
           Should diagonal lines be treated as a slope?
    • list polar.pointList = []
```

Trajectory points - will only be initialized in main if usingPointList is set to true.

polar.veclen = lambda p,q: sqrt(sum([(u-v)\*\*2 for u,v in zip(p,q)]))

Length of vector from point p = (x1,y1) to q = (x,y).

68 File Documentation

```
polar.subvec = lambda p0,p1: [(i-j) for i,j in zip(p1,p0)]
      Return vector p1-p0.

    polar.dotprod = lambda p0,p1,p: sum([u*v for u,v in zip(subvec(p0,p1),subvec(p1,p))])

      Dot product of vectors from point p0=(x0,y0) to p1=(x1,y1) and from p1=(x1,y1) to p=(x,y).

    polar.clamp = lambda a: min(max(-1,a),1)

      Clamp a value to range [-1,1].

    polar.cartesian2Polar = lambda x,y: (sqrt(x*x+y*y), atan2(-y,-x)+pi)

      Get the polar coordinates from cartesian coordinates.

    polar.setColor = lambda v: joe.color("red" if v < 0 else "blue")</li>

      Set the color of the curve.
• int polar.wrapPi = lambda x: x - 360 * ((x + 180) // 360)
      Wraps an angle to \pm 180.

    bool polar.__toDebug__ = False

      Toggle debugging mode.
• int polar.NBITS = 8
      Number of bits after binary point.

    int polar.BSCALE = 2**NBITS

      Scale for fixed point: 256 or 65536, for instance.

    polar.float2Int = lambda f: int(f*BSCALE)

      Maps a float number to integer.

    polar.int2Float = lambda b: ldexp(float(b),-NBITS)

      Maps an integer number to float.

    list polar.bam_bit_table = [ 0.0055, 0.0109, 0.0219, 0.0439, 0.088, 0.1757, 0.3515, 0.703, 1.406, 2.8125, 5.625,

  11.25, 22.5, 45.0, 90.0, 180.0]
      BAM bit table.
int polar.WSIZE = len(bam_bit_table)-1
      Word size for BAM.
• int polar.LSB = 2**-WSIZE * 180
      Least Significant Bit.
• int polar.f1 = 4: (a * 2*sin(n*t), t)
      A rose of n or 2n petals.
• int polar.f2 = lambda a, t: f1(a,t,5)
      A rose of five petals.
• int polar.f3 = lambda a, t: f1(a,t,3)
      A rose of three petals.

    int polar.f4 = lambda a, t: f1(a,t,2)

      A rose of four petals.
• polar.f5 = lambda a, t: (a * cos(t/2.0), t)
      Double Loop.

    polar.f7 = lambda a, t: (a * 2 * sin(t), t)

      Circle.
• polar.f8 = lambda a, t: (a * (2*sin(2*t)+1), t)
• polar.f9 = lambda a, t: (a * (\cos(5*t)**2 + \sin(3*t) + 0.3), t)
      Oscar's butterfly.
• polar.f10 = lambda a, t: (a * (\sin(t) + \sin(5*t/2)**3), t)
```

Crassula Dubia.

List of tuples with: polar equation function, angle range, initial angle, title and number of segments.

# .3 turtle.py File Reference

#### Classes

· class turtle.FlailDriver

#### **Namespaces**

turtle

# **Functions**

- def turtle.title (s)
- def turtle.setup (width, height)
- def turtle.bgcolor (r, g, b)
- def turtle.window\_width ()
- def turtle.window\_height ()
- def turtle.setworldcoordinates (x0, y0, x1, y1)
- def turtle.speed (v)

70 File Documentation

# Index

del_	polar, 36
flailDriver::FlailDriver, 52	
turtle::FlailDriver, 58	descend
init	flailDriver::FlailDriver, 53
flailDriver::FlailDriver, 52	turtle::FlailDriver, 59
turtle::FlailDriver, 58	dot
toDebug	flailDriver::FlailDriver, 53
polar, 34	turtle::FlailDriver, 59
_toDenary	dotprod
polar, 15	polar, 36
_toDenaryRec	drawAxis
polar, 15	polar, 17
	drawBox
ascend	polar, 18
flailDriver::FlailDriver, 52	,
turtle::FlailDriver, 58	f
ascension	flailDriver::FlailDriver, 56
polar, 15	turtle::FlailDriver, 62
axes	f0
polar, 16	polar, 18
	f1
BAM2float	polar, 36
polar, 16	f10
BSCALE	polar, 37
polar, 35	f11
backward	polar, 37
flailDriver::FlailDriver, 52	f12
turtle::FlailDriver, 59	polar, 19
bam_bit_table	f13
polar, 35	polar, 38
bgcolor	f14
flailDriver, 9	polar, 19
turtle, 47	f15
	polar, 20
cartesian2Polar	f16
polar, 35	polar, 39
clamp	f17
polar, 35	polar, 39
color	f18
flailDriver::FlailDriver, 53	polar, 39
turtle::FlailDriver, 59	f19
crossprod	polar, 21
polar, 17	f2
crossprod3	polar, 40
polar, 17	f20
curveList	polar, 22

72 INDEX

f21		right, 55
	polar, 22	setheading, 55
f22	1 40	setposition, 56
(00	polar, 40	write, 56
f23		float2BAM
10.4	polar, 23	polar, 25 float2Int
f24		
40E	polar, 23	polar, 42 float2UBAM
f25	polor 04	polar, 26
40	polar, 24	formati
f3	polor 40	flailDriver::FlailDriver, 56
f4	polar, 40	turtle::FlailDriver, 63
14	polar, 41	formatu
f5	polal, 41	flailDriver::FlailDriver, 57
15	polar, 41	turtle::FlailDriver, 63
f6	polal, 41	forward
10	polar, 24	flailDriver::FlailDriver, 53
f7	polar, 24	turtle::FlailDriver, 60
.,	polar, 41	·
f8	polar, Ti	goto
	polar, 42	flailDriver::FlailDriver, 54
f9	po.a.,	turtle::FlailDriver, 60
	polar, 42	LIEIOLIT
flail[	Oriver, 9	HEIGHT
	bgcolor, 9	polar, 43
		help
	setup, 9	polar 20
	setup, 9 setworldcoordinates, 10	polar, 28
	setworldcoordinates, 10	home
	•	home flailDriver::FlailDriver, 54
	setworldcoordinates, 10 speed, 10	home
	setworldcoordinates, 10 speed, 10 title, 10	home flailDriver::FlailDriver, 54
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60
	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60 initPointList
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:FlailDriverdel_, 52	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:FlailDriver	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver::FlailDriverdel_, 52init, 52 ascend, 52	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver.py, 65 Driver::FlailDriverdel, 52init, 52 ascend, 52 backward, 52	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43 left
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver.py, 65 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:py, 65 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver.py, 65 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:py, 65 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:PlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formatu, 57 forward, 53 goto, 54	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53 goto, 54 home, 54	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44  main
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver.py, 65 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53 goto, 54 home, 54 left, 54	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44  main polar, 29
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:FlailDriverdel, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53 goto, 54 home, 54 left, 54 pendown, 54	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44  main polar, 29  move polar, 29
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53 goto, 54 home, 54 left, 54 pendown, 54 pensize, 54	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44  main polar, 29  move polar, 29  NBITS
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver:py, 65 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53 goto, 54 home, 54 left, 54 pendown, 54 pensize, 54 penup, 55	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44  main polar, 29  move polar, 29
flail[	setworldcoordinates, 10 speed, 10 title, 10 window_height, 10 window_width, 10 Driver.FlailDriver, 51 Driver::FlailDriverdel_, 52init, 52 ascend, 52 backward, 52 color, 53 descend, 53 dot, 53 f, 56 formati, 56 formatu, 57 forward, 53 goto, 54 home, 54 left, 54 pendown, 54 pensize, 54	home flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  initPointList polar, 28 int2Float polar, 43  LSB polar, 43  left flailDriver::FlailDriver, 54 turtle::FlailDriver, 60  LH polar, 43  LW polar, 44  main polar, 29  move polar, 29  NBITS

INDEX 73

flailDriver::FlailDriver, 54	float2Int, 42
turtle::FlailDriver, 60	float2UBAM, 26
pensize	HEIGHT, 43
flailDriver::FlailDriver, 54	help, 28
turtle::FlailDriver, 61	initPointList, 28
penup	int2Float, 43
flailDriver::FlailDriver, 55	LSB, 43
turtle::FlailDriver, 61	LH, 43
pointList	LW, 44
polar, 44	main, 29
polar, 11	move, 29
toDebug, 34	NBITS, 44
_toDebag, 64 _toDenary, 15	pointList, 44
_toDenaryRec, 15	polar2Cartesian, 30
ascension, 15	polarRose, 30
axes, 16	setColor, 44
BAM2float, 16	staircase, 45
BSCALE, 35	subvec, 45
bam bit table, 35	toBinary, 32
cartesian2Polar, 35	toDenary, 32
clamp, 35	toFloat, 32
crossprod, 17	toInt, 33
crossprod3, 17	toUBAM, 33
curveList, 36	updateBBOX, 34
dotprod, 36	usingFlail, 45
•	veclen, 45
drawPay 19	WIDTH, 46
drawBox, 18	WSIZE, 46
f0, 18	wrapPi, 46
f1, 36	Xc, 46
f10, 37	Yc, 47
f11, 37	polar.py, 65
f12, 19	polar2Cartesian
f13, 38	polar, 30
f14, 19	polarRose
f15, 20	polar, 30
f16, 39	p 0.50., 0 0
f17, 39	repeat
f18, 39	flailDriver::FlailDriver, 55
f19, 21	turtle::FlailDriver, 61
f2, 40	reset
f20, 22	flailDriver::FlailDriver, 55
f21, 22	turtle::FlailDriver, 61
f22, 40	right
f23, 23	flailDriver::FlailDriver, 55
f24, 23	turtle::FlailDriver, 61
f25, 24	
f3, 40	setColor
f4, 41	polar, 44
f5, 41	setheading
f6, 24	flailDriver::FlailDriver, 55
f7, 41	turtle::FlailDriver, 62
f8, 42	setposition
f9, 42	flailDriver::FlailDriver, 56
float2BAM, 25	turtle::FlailDriver, 62

74 INDEX

setup	penup, 61
flailDriver, 9	repeat, 61
turtle, 48	reset, 61
setworldcoordinates	right, 61
flailDriver, 10	setheading, 62
turtle, 48	setposition, 62
speed	write, 62
flailDriver, 10	
turtle, 48	updateBBOX
staircase	polar, 34
polar, 45	usingFlail
subvec	polar, 45
polar, 45	veclos
	veclen
title	polar, 45
flailDriver, 10	WIDTH
turtle, 48	polar, 46
toBinary	WSIZE
polar, 32	polar, 46
toDenary	window_height
polar, 32	flailDriver, 10
toFloat	turtle, 49
polar, 32	window_width
toInt	flailDriver, 10
polar, 33	turtle, 49
toUBAM	wrapPi
polar, 33	polar, 46
turtle, 47	write
bgcolor, 47	flailDriver::FlailDriver, 56
setup, 48	turtle::FlailDriver, 62
setworldcoordinates, 48	<b>V</b>
speed, 48	Xc
title, 48	polar, 46
window_height, 49	Yc
window_width, 49	polar, 47
turtle.FlailDriver, 57	ροιαί, 47
turtle.py, 69	
turtle::FlailDriver del , 58	
uei_, 58 init, 58	
ascend, 58	
backward, 59	
color, 59	
descend, 59	
dot, 59	
f, 62	
formati, 63	
formatu, 63	
forward, 60	
goto, 60	
home, 60	
left, 60	
pendown, 60	
pensize, 61	