Week 1

Evaluation, Variables and Turtle

Python Shell (Console)

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC
v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> 5 + 3
8
>>> 'M' + 'iss' * 2 + 'ppi'
'Mississppi'
>>> x = 5 + 3
>>> x
8
>>>
```

Echoes: “return values” of the previous line

But sometimes there will NOT be any echo as there is NO “return value”

Ln: 10 Col: 4

- However, this should NOT be the main area we work in (i.e. 90% of the time)

Arithmetic Evaluation

- What will be the evaluated values for the following:

  3 * 4 + 5

  3 + 4 * 5

  5 ** 3 % 4

  97 / 4

  97 // 4

You should try evaluating these WITHOUT typing into Python first

Logical Evaluation

- What will be the evaluated values for the following:

  1 == 1

  3 + 2 == 1 + 4

  3 + 2 != 1 + 4

  4 > 3

  4 > 4

  6 + 3 < 9 + 3

  True or False

  True and (False or True)

Logical Evaluation

- What will be the evaluated values for the following:

  not True

  not False

  not not True

  not 0

  not 9999

  0 and 9999

  not ‘abc’

  not ‘’

More about Truth Values

- Python has keywords `True` and `False`
- In Python 3.x, `True` and `False` will be equal to `1` and `0` respectively
- Anything that is **not** zero or empty or `None` will be evaluated as `True`
- Logic:

```
>>> True and 0
0
>>> not 'abc'
False
>>> 1 or 0
1
```

I-don’t-know Evaluation

- What will be the evaluated values for the following:

  True + 1

  False * 5

  0 + (not 1)

String Evaluation

- What will be the evaluated values for the following:

```
'abc' + 'def'
'gala' * 3
'mu' + 'ha' * 4
('ba '*2+'bidu'*2+'bi ' + 'jam '*2)*3

'banana'[3]
'banana'[2:4]
'banana'[1::2]
```

String Slicing

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- Let s = ‘abcdef’
- What is the result of `s[2]` and `s[2:]` and `s[2::]`?
  - Are they the same?
- Only `s[2:]` and `s[2::]` are the same.
- What happens if we do `s[1:1]`?
  - Get ‘’ (a blank string)

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- Let s = ‘abcdef’
- What is the result of `s[]` and `s[:2]` and `s[:2:]`?
  - Are they the same?

- Only `s[:2]` and `s[:2:]` are the same.
- `s[]` is a syntax error

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- Let s = ‘abcdef’
- What about `s[5:0:-1]`?

‘`fedcb`’

- What happens if we do `s[:2:-1]`?

‘`fed`’

- Lecture example: `s[::-1]`

‘`fedcba`’

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

String Slicing

	a	b	c	d	e	f
Index	0	1	2	3	4	5

- Let s = ‘abcdef’
- What happened? By Python convention, if step is **negative**, default start is the last letter and default stop is the first letter, inclusive.

- Lecture example: s[::-1] is interpreted as “reversing the string”

‘fedcba’

- So is s[::-1] the same as s[5:-1:-1]?
- No. s[5:-1:-1] will return a blank string.

Start – By default, start from index 0.
Stop – By default, include the last letter.
Step – By default, “jump” by 1 step.

Default

- If step > 0
  - Start – By default, start from index 0.
  - Stop – By default, include the last letter.
  - Step – By default, “jump” by 1 step.
- Else (step < 0)
  - Default start = last letter
  - Default end = -n-1

- Let n = length of your string
- If step > 0
  - Start = 0
  - Stop = n
- Else if step < 0
  - Start = n-1
  - Stop = -n-1

ASCII Table

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38					70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39					71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40					72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41					73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T				t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	&#					u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	&#					v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	&#					w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	&#					x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

'K' is greater than 'D'

'p' is greater than 'P'

Lexicographical Order

- Compare letter-by-letter from left to right

```
>>> 'abc' > 'abd'
False
>>> 'abc' > 'aba'
True
```

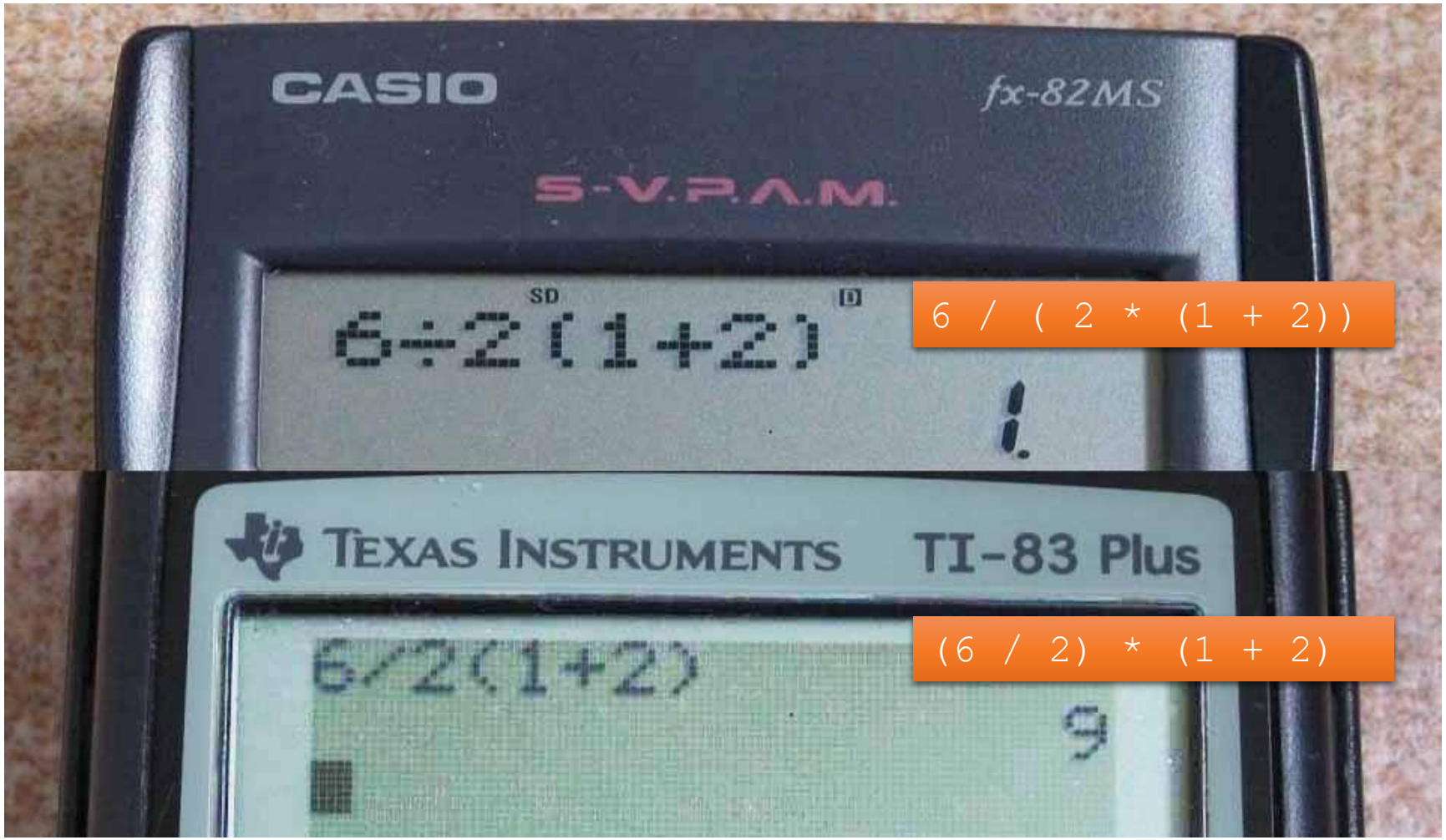
- “winner” is decided when the comparison reaches a different letter

```
>>> 'ab' > 'aaabbbcccddd'
True
```

- “anything” is greater than “nothing”

```
>>> 'abcd' > 'abc'
True
```

Operator Precedence



Python Operator Precedence

- 6/2*(1+2)
- 6/2*(1+2)
- 6/2*3
- 3*3
- 9

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'td>
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

How Do I Remember It All ... ? BODMAS !

B	**B**rackets first
O	**O**rders (i.e. Powers and Square Roots, etc.)
DM	**D**ivision and **M**ultiplication (left-to-right)
AS	**A**ddition and **S**ubtraction (left-to-right)

Divide and Multiply rank equally (and go left to right).

Add and Subtract rank equally (and go left to right)

Arithmetic Evaluation

- What will be the evaluated values for the following (or what is the orders of the operators?)

  1 + 2 * 3

  1 + 2 * 3 **4

  1 + 2 * 3 **4 – 5

  not 0 + 1

What is the difference?

- What do we have when we ask if 1 is it the same as ‘1’?

  `1 == ‘1’`

- Or what is the difference between the following two lines?

  `123+456`

  `‘123’+‘456’`

Type Conversions

```
>>> type(123)
<class 'int'>
>>> 123 + 456
579
>>> type('123')
<class 'str'>
>>> '123' + '456'
'123456'
>>> '123' + 456
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    '123' + 456
TypeError: can only concatenate str (not "int") to str
```

Note that the “+” operator performs differently for different types

Type Conversions

- Converting floats into integers

```
>>> int(1.234)
1
>>> int(1.7)
1
```

- Can I say the integer produced is always smaller than the input float?

```
>>> int(-2.3)
-2
```

Variables

- Now you should know the following:

  3 * 4 + 5

  3 + 4 * 5

- How about

  x = 3

  y = 4

  z = 5

  x * y + z

Variables Can Store Any Type

```
>>> 5 > 3
True
>>> x = 5 > 3
>>> x
True
>>> 5 > 3 and 3 > 9
False
>>>
```

“Creation” of Variables

- What will be the evaluated values for the following:

  a * b + c

- Error! Why?
- Because a, b and c are undeclared
  - In another words, “not created”, “not born yet”
  - Whenever you type a line

    a = …. (something)
  - A variable (a) is born

From scratch

m + 3

- Error!

m = 1

m + 3

- Output:

4

Creation of the variable m

Turtle Graphics

```
>>> from turtle import *
>>> forward(100)
>>>
```

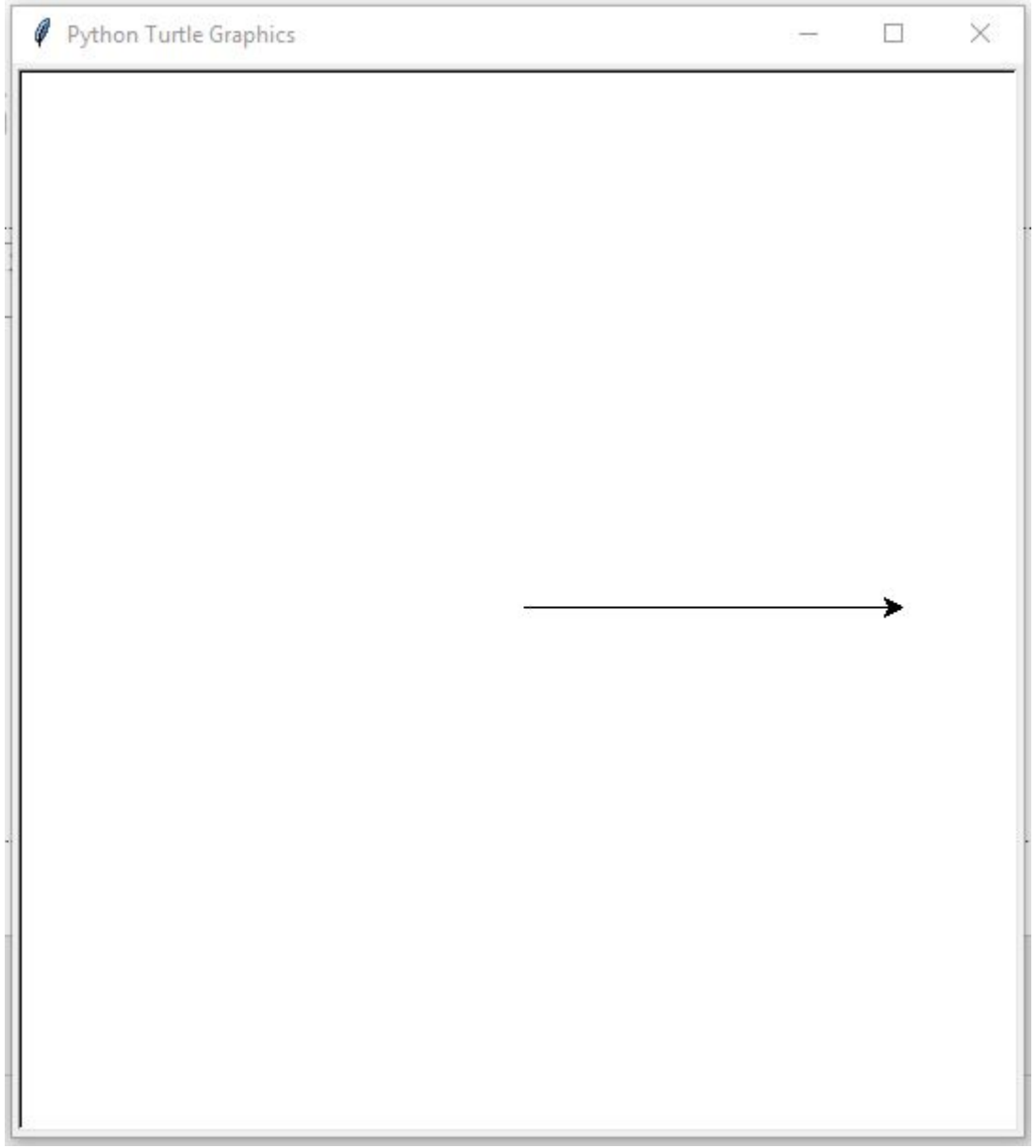
- Or you can use the short from

```
>>> from turtle import *
>>> fd(100)
```

- Or this,

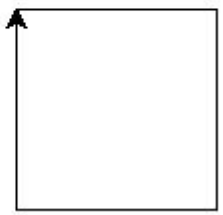
```
>>> import turtle
>>> turtle.fd(100)
```

- But for our course, please do NOT use the last form



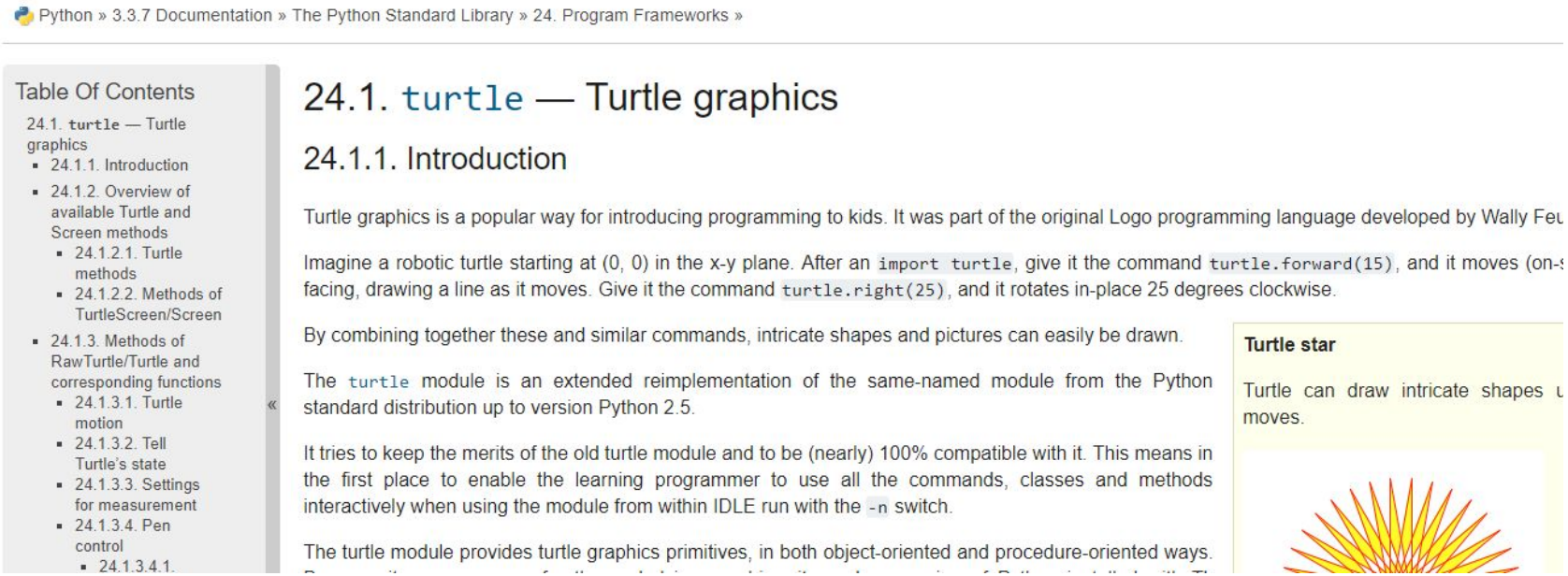
Turtle Graphics

```
>>> from turtle import *
>>> fd(100)
>>> rt(90)
>>> fd(100)
>>> rt(90)
>>> fd(100)
>>> rt(90)
>>> fd(100)
>>>
```



More Turtle Commands

- You can go to the website: https://docs.python.org/3.3/library/turtle.html?highlight=turtle
- Or just google “Python Turtle”



Python » 3.3.7 Documentation » The Python Standard Library » 24. Program Frameworks »

Table Of Contents

24.1. turtle — Turtle graphics

24.1.1. Introduction

Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feu

Imagine a robotic turtle starting at (0, 0) in the x-y plane. After an `import turtle`, give it the command `turtle.forward(15)`, and it moves (on-s facing, drawing a line as it moves. Give it the command `turtle.right(25)`, and it rotates in-place 25 degrees clockwise.

By combining together these and similar commands, intricate shapes and pictures can easily be drawn.

The `turtle` module is an extended reimplementation of the same-named module from the Python standard distribution up to version Python 2.5.

It tries to keep the merits of the old turtle module and to be (nearly) 100% compatible with it. This means in the first place to enable the learning programmer to use all the commands, classes and methods interactively when using the module from within IDLE run with the `-n` switch.

The turtle module provides turtle graphics primitives, in both object-oriented and procedure-oriented ways.

Turtle star

Turtle can draw intricate shapes u moves.

- Actually, this is what most programmers do