PyLadies Berlin @ Immobilienscout24
Turtorial

Stefan Nordhausen

Platform Engineering

Feb 15 2017

Plan for the Evening

- I talk, you code.
- If you have questions, feel free to ask.
- Code examples/solutions are on https://github.com/ImmobilienScout24/turtorial
- Topics covered
  - define a function
  - for-loops
  - reusability
  - code readability / style
  - recursion
  - Python 2 vs. Python 3
  - How to teach Python

About me

- Started programming >20 years ago
- 10 years experience in software development
- Started with Python 8 years ago
- What I do at Immobilienscout
  - Infrastructure automation with Python
  - Python help/training for collegues

What's a Turtorial?

- Turtorial: Tutorial that uses the turtle module
- Module to learn the basics of programming
- Idea: A turtle walking around
  - When it moves, it leaves a line in the sand
  - You can tell it to move or to turn
- Completely useless for actual work
- Great for teaching programming, thanks to visual feedback

Turtle Preconditions

- Text editor / IDE of your choice
- Python interpreter
- Turtle module installed

- Ubuntu
  - sudo apt-get install python-tk
  - create a virtualenv

- MacOSX
  - sudo port install py27-tkinter

Turtle Basics

```
>>> import turtle
>>> turtle.forward(100)
>>> turtle.left(90)
>>> turtle.forward(80)
>>> turtle.left(45)
>>> turtle.forward(80)
```

Turtle More Basics

- Undo the last action
  - You can undo multiple times

```
turtle.undo()
```

- Undo everything

```
turtle.reset()
```

Turtle Exercise 1

- Draw a square!
- Hint 1: Do NOT call your file turtle.py!
- When you "import turtle", Python will think you want to import your own module.
- Hint 2: When the program terminates, your square disappears
- Add raw_input() or input() at the end of your program
- Advanced Pythonistas: Help the beginners

Turtle Exercise 1

- Draw a square!
- Solution:
  - Code is in turtorial_01_square.py

```
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
```

Turtle Exercise 1

- This code is ugly
  - Lots of copy & paste
  - If you want a bigger/smaller square: more copy & paste
  - Not re-usable (unless you copy & paste)

Turtle Exercise 2

- Write a function that draws a square!
  - The size of the square should be a parameter

```
# Functions in Python are defined like this:
def my_useful_function(parameter1, parameter2):
    command_one()
    command_two()

# Call the function like this:
my_useful_function(123, "the second parameter")
```

Turtle Exercise 2

- Solution:
  - Code is in turtorial_02_square.py

```
def draw_square(size=100):
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
    turtle.forward(size)
    turtle.left(90)
```

- "size=100" defines a default value for the parameter
  - If you call without parameter, the default is used
  - If you call with a parameter, the parameter is used

For-loops and range()

- Our code is still very repetitive
- We actually want something like this:

```
please do this 4 times:
    turtle.forward(size)
    turtle.left(90)
```

For-loops and range()

- Python (and most other programming languages) have "for" loops

```
for x in range(10):
    print x
```

- You get a list with the numbers 0...9
  - It's not 1...10
- The loop runs once for each item in the list

Turtle Exercise 3

- Rewrite the function to use a for-loop

Turtle Exercise 3

- Solution:
  - Code is in turtorial_03_square.py

```
def draw_square(size=100):
   for x in range(4):
        turtle.forward(size)
        turtle.left(90)
```

Excursion: Code readability

- Some people says: The code was hard to write, so it should be hard to read.
- Practice shows
  - Code is read much more often than it is written
  - The person reading the code is probably you, 1 year later
  - When other people cannot read your code, they constantly ask you
- Do yourself a favor and produce code that is easy to read

Excursion: Code readability

General rules for all programming languages:

- Choose good names for your variables, functions etc.
  - What is better: square() or draw_square()?
- Names, comments and documentation in English
  - British or American?
- Keep your functions small and focussed
  - Should do one thing
  - functions of more than 30 lines are most likely too large
- Many more

Excursion: Code readability

Rules specific to Python

- Python is standardized using PEPs (Python Enhancement Proposals)
- PEP-8 defines coding style
  - Not everybody follows it to the very last letter
  - Some built-in modules of Python do not follow it, because they are older than PEP-8
- Why a style guide?
  - Consistent formatting makes code easier to read for others
  - Tools that create documentation based on your code rely on certain format

Excursion: Code readability: Tools

Three tools you should know:

- flake8 checks for PEP-8 compliance
  - it is incredibly picky
  - it is never wrong
- autopep8 automatically formats Python code to conform to the PEP 8 style guide
- pylint checks for general best practices / common errors
- Install them with

```
pip install flake8 autopep8 pylint
```

Excursion: Code readability: Tools

- For the previous solution, flake8 says:

```
$ flake8 turtorial_03_square.py
turtorial_03_square.py:5:1: E302 expected 2 blank lines, found 1
```

- pylint says:

```
C:  1, 0: Missing module docstring (missing-docstring)
C:  5, 0: Missing function docstring (missing-docstring)
C:  6, 8: Invalid variable name "x" (invalid-name)
E:  7, 8: Module 'turtle' has no 'forward' member (no-member)
E:  8, 8: Module 'turtle' has no 'left' member (no-member)
W:  6, 8: Unused variable 'x' (unused-variable)

[...]
Global evaluation
-----------------
Your code has been rated at -7.50/10 (previous run: -7.50/10, +0.00)
```

Excursion: Code readability: Tools

- Fixing all the complaints (see turtorial_04_square.py)

```
"""A version of turtorial_03_square.py that flake8 and pylint like"""
import turtle


def draw_square(size=100):
    """Draw a square, default size is 100."""
    for _ in range(4):
        # Disable error checking for a single statement:
        turtle.forward(size) # pylint: disable=E1101

        # Disable error checking for entire block.
        # pylint: disable=E1101
        turtle.left(90)


draw_square()
raw_input()
```

Excursion: Code readability: Tools

Further reading:

- https://www.python.org/dev/peps/pep-0008/
- https://www.pylint.org/
- https://flake8.readthedocs.io
- https://github.com/hhatto/autopep8
- Get style-check-as-a-service, e.g. from www.landscape.io
  - https://landscape.io/github/ImmobilienScout24/afp-cli/
- And now, back to turtle

Turtle Exercise 4

- Write a function that can draw a polygon
  - The size should be a parameter
  - The number of sides should be a parameter
- Hints:
  - Start with a copy of your square() function
  - In total, you need to turn 360 degrees
    - so after each side, you need to turn ... degrees

Turtle Exercise 4

- Solution:
  - Code is in turtorial_05_polygon.py

```
import turtle

# Draw a polygon, default is a square of size 100.
def draw_polygon(length=100, sides=4):
    for side in range(sides):
        turtle.forward(length)
        turtle.left(360 / sides)
```

- This works
  - 3, 4, 5, 6 sides
  - 7 sides looks suspicious

Primary School Math

- By default, you get integer division

```
>>> 360 / 7
51
>>> 360 % 7
3
>>> 51 * 7
357
>>> 51 * 7 + 3
360
```

Primary School Math

- If you want floating point math, you need floating point parameters

```
>>> # This does NOT work:
>>> float(360 / 7)
51.0

>>> # This works:
>>> float(360) / 7
51.42857142857143
>>> 360 / 7.0
51.42857142857143
```

Primary School Math

- This is different in Python 3

```
$ python3.4
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
>>> 360 / 7
51.42857142857143
```

Primary School Math

- How to write programs that work for Python 2 and Python 3?
- "from __future__" imports bring Python 3 changes to Python 2

```
from __future__ import division, print_function

# Will always print 51.42... thanks to __future__ import
print(360 / 7)

# Will always print 51
print(360 // 7)
```

- See turtorial_06_polygon.py for code example

Recursion

- A recursive function is a function that calls itself
  - usually many times in a row
- Used for
  - freaky algorithms (quicksort, B-trees...)
  - file systems (grep -r, chmod -R)

Recursion

Typical elements of recursive functions:

- There must be a condition that eventually stops the recursion
- The function modifies the parameters before calling itself
  - Otherwise, recursion would never stop
- When I explain quicksort, half the audience is confused
  - "Too abstract!"
  - "Too theoretical!"

Recursion

Let's modify the polygon function instead:

```
def draw_polygon(length=100, sides=4):
    for side in range(sides):
        turtle.forward(length)
        turtle.left(360 / sides)
        if length > 10 and sides > 3:
            draw_polygon(length=length / 3, sides=sides - 1)
```

- See turtorial_08_recursion.py for full code

Haus vom Nikolaus

- Write a function that draws Das Haus vom Nikolaus
  - Wait, what?
  - Game/riddle for children
  - https://de.wikipedia.org/wiki/Haus_vom_Nikolaus

Haus vom Nikolaus

- A simple children's game
- Turns out to be surprisingly complex:
  - How long are the diagonals?
  - What angle do we use for the roof?
  - How long do the lines of the roof need to be?
  - You have to get all the turn() commands right

Haus vom Nikolaus

- How long are the diagonals?
  - The diagonals produce triangles with 90° angles -> Pythagoras
  - $a^2 + b^2 = c^2$
  - diagonal = (2 * size**2) ** .5
- What angle do we use for the roof?
  - 45°
- How long do the lines of the roof need to be?
  - If you used 45° above, their length is diagonal / 2

Haus vom Nikolaus

See turtorial_09_nikolaus.py for full code.

```
def hvn(size=100):
    diagonal = (2 * size**2) ** .5
    turtle.forward(size)
    turtle.left(135)
    turtle.forward(diagonal)
    turtle.right(90)
    turtle.forward(diagonal / 2)
    turtle.right(90)
    turtle.forward(diagonal / 2)
    turtle.right(90)
    turtle.forward(diagonal)
    turtle.right(135)
    turtle.forward(size)
    turtle.right(90)
    turtle.forward(size)
    turtle.right(90)
    turtle.forward(size)
```

Haus vom Nikolaus

- Lesson learned: Don't underestimate complexity.

Haus vom Nikolaus

- Another solution
  - Analyze the task
    - Haus des Nikolaus consists of 2 squares + 2 lines
  - Re-use the square function

```
def hvn(size=100):
    """Draw Das Haus vom Nikolaus, using draw_square()"""
    half_diagonal = (2 * size**2) ** .5 / 2

    draw_square(size)
    # Draw a "fish".
    turtle.left(45)
    turtle.forward(half_diagonal)
    draw_square(size=half_diagonal)
    turtle.right(90)
    turtle.forward(half_diagonal)
```

Thank's

- Thanks for coming
- Questions?
- Contact
  - stefan.nordhausen@scout24.com
  - https://github.com/snordhausen