



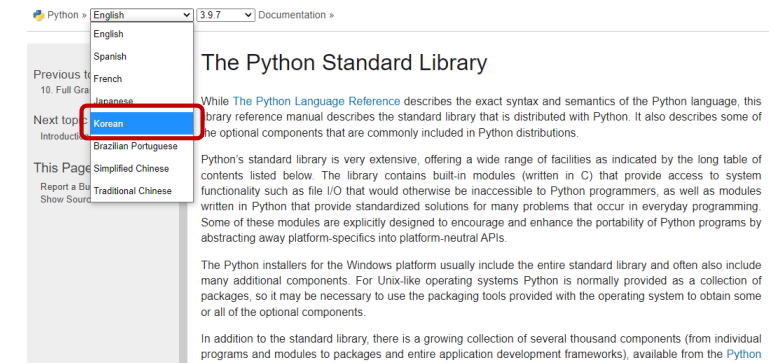
Python: Standard Library

Sunglok Choi, Assistant Professor, Ph.D.
Computer Science and Engineering Department, SEOULTECH
sunglok@seoultech.ac.kr | <https://mint-lab.github.io/>

Python: Basic → Intermediate + Standard Library

- Data Types
- Operators
- Flow Control
- Function Definition
- Object-oriented Programming
- File Input and Output
- Exception Handling
- Package Import
- The Python Standard Library / 파이썬 라이브러리 (Wikidocs)
 - math, decimal, random
 - time / glob, fnmatch
 - csv, pickle
 - tkinter / turtle

Image: [Python Official Documentation](#), [Wikipedia](#)



Tip for Korean students!

Article | Talk | Read | View source | View history | Search Wikipedia

Pi

From Wikipedia, the free encyclopedia

This article is about the mathematical constant. For the Greek letter, see [Pi \(letter\)](#). For other uses, see [Pi \(disambiguation\)](#).

The number [π](#) (/ˈpaɪ/; spelled out as "pi") is a mathematical constant, approximately equal to 3.14159. It is defined in Euclidean geometry^[a] as the ratio of a circle's circumference to its diameter, and also has various equivalent definitions. The number appears in many formulas in all areas of mathematics and physics. The earliest known use of the Greek letter π to represent the ratio of a circle's circumference to its diameter was by Welsh mathematician William Jones in 1706.^[1] It is also referred to as **Archimedes's constant**.^{[2][3][4]}

Being an irrational number, π cannot be expressed as a common fraction, although fractions such as 22⁄7 are commonly used to approximate it. Equivalently, its decimal representation never ends and never settles into a permanently repeating pattern. Its decimal (or other base) digits appear to be randomly distributed, and are conjectured to satisfy a specific kind of statistical randomness.

It is known that π is a transcendental number;^[3] it is not the root of any polynomial with rational coefficients. The transcendence of π implies that it is impossible to solve the ancient challenge of squaring the circle with a compass and straightedge.

Ancient civilizations, including the Egyptians and Babylonians, required fairly accurate approximations of π for practical computations. Around 250 BC, the Greek mathematician Archimedes created an algorithm to approximate π with arbitrary accuracy. In the 5th

Part of a series of articles on the mathematical constant π

0	1	2	3	4
			π	
3.14159 26535 89793 23846 26433...				
Uses				
Area of a circle · Circumference · Use in other formulae				
Properties				
Irrationality · Transcendence				
Value				
Less than 22/7 · Approximations · Memorization				
People				
Archimedes · Liu Hui · Zu Chongzhi · Aryabhata · Madhava · Ludolph van Ceulen · Seki Takakazu · Takebe Kenko · William Jones · John Machin · William Shanks · Srinivasa Ramanujan · John Wrench · Chudnovsky brothers · Yasumasa Kanada				
History				
Chronology · Book				
In culture				
Legislation · Pi Day				
Related topics				
Squaring the circle · Basel problem · Six nines in π · Other topics related to π				

⋮ · 🔍 · E

My Comments for Better Python Programming



1. Take advantages of Python itself (a.k.a. *Pythonic*)

- e.g. Swap using unpacking

```
temp = a
a = b    VS.    (a, b) = (b, a)
b = temp
```

- References
 - [Code Style](#), The Hitchhiker's Guide to Python
 - [Write More Pythonic Code](#), Real Python
 - [PEP 8 – Style Guide for Python Code](#), Python

2. Utilize the exiting libraries (a.k.a. [Don't reinvent the wheel](#)) and master them if they are useful

- Problem #1) Too many libraries
 - Search your keywords in **Google**/[Github](#) (with *python*), [PyPI](#), and ...
- Problem #2) A few documents and examples
 - Select a popular one (if possible)
 - Search your problem in **Google** (or read and analyze the source codes)

math: Mathematical Functions

- math provides various and useful mathematical functions similar to the C standard library.
- API examples
 - Constants: `pi` (3.14..., the ratio of a circle's circumference to its diameter; 원주율), `e` (2.718..., the base for natural logarithm; 자연상수), `inf` (a floating-point positive infinity), `nan` (a floating-point not-a-number; NaN)
 - `sqrt(x)`: Return the square root of x
 - `log(x[, base])`, `log2(x)`, `log10(x)`: Return the natural logarithm, base-2 logarithm, and base-10 logarithm of x
 - `ceil(x)`: Return the smallest integer greater than or equal to x
 - `prod(iterable, *, start=1)`: Calculate the product of all the elements in the input `iterable`
 - `sin(x)`, `cos(x)`, `tan(x)`, ..., `atan(x)`, `atan2(y, x)`: Trigonometric functions
 - `degrees(x)`, `radians(x)`: Convert angle x from radians to degrees (or vice versa)
 - `isinf(x)`, `isnan(x)`, `isfinite(x)`: Check whether x is an infinity or a NaN or finite (neither an infinity nor a NaN)
- Usage example

```
import math
factorial_prod = lambda n: math.prod(range(1, n + 1))
print(factorial_prod(10)) # 3628800
```

decimal: Decimal Fixed-point and Floating-point Arithmetic

- decimal provides fast correctly-rounded decimal floating-point arithmetic.
- API examples
 - `Decimal`: A class for a decimal floating-point value
 - `Decimal.quantize(exp, rounding=None, context=None)`: Return a rounded value with precision of `exp`

- Usage example: Rounding

Using the default built-in function

```
print(round(3.5)) # 4
```

```
print(round(4.5)) # 4 (not 5)
```

Using the decimal module

```
import decimal
```

```
print(decimal.getcontext()) # rounding=decimal.ROUND_HALF_EVEN
```

```
print(decimal.Decimal(3.5).quantize(1, decimal.ROUND_HALF_UP)) #
```

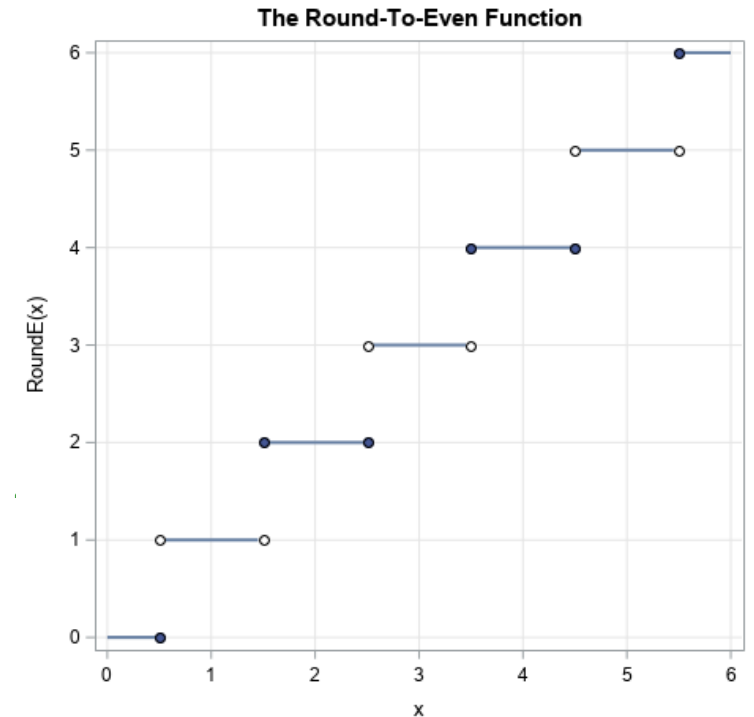
```
print(decimal.Decimal(4.5).quantize(1, decimal.ROUND_HALF_UP)) #
```

Using a simple hand-made function

```
round2 = lambda x: int(x + 0.5)
```

```
print(round2(3.5)) # 4
```

```
print(round2(4.5)) # 5
```



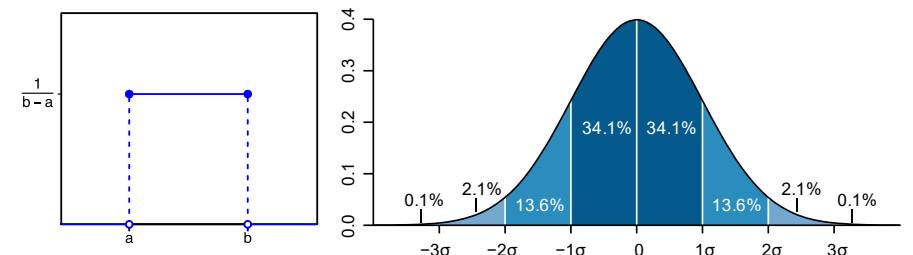
Why? To avoid systemic bias

e.g. `round(1.5) + round(2.5) +`
`round(3.5) + round(4.5)`

random: Pseudo-random Number Generators

- random provides pseudo-random generators for various probability distributions.
- API examples
 - `random()`: Return the next random floating-point number in the range [0.0, 1.0)
 - `randint(a, b)`: Return a random integer N such that $a \leq N \leq b$
 - `uniform(a, b)`: Return a random floating-point number N such that $a \leq N \leq b$
 - `gauss(mu, sigma)`, `normalvariate(mu, sigma)`: Return a random floating-point number under Gaussian distribution whose mean is mu and standard deviation is sigma
 - `seed(a=None, version=2)`: Initialize the random number generator (a=None; the current system time is used)
- Usage example: Uniform vs. Gaussian

```
import random
round2 = lambda x: int(x + 0.5)
print([round2(random.uniform(0, 10)) for i in range(10)]) # [2, 7, 6, 0, 3, 5, 3, 5, 7, 1]
print([round2(random.gauss(5, 1)) for i in range(10)])   # [5, 6, 6, 3, 5, 5, 5, 5, 7, 6]
```



time: Time Access and Conversions

- [time](#) provides various time-related functions. (For related functionality, see also [datetime](#) and [calendar](#))
- API examples
 - `time()`: Return the current time in seconds [since the epoch](#) (January 1st, 1970, 00:00:00 ([UTC](#)) for Unix)
 - `process_time()`, `thread_time()`: Return the sum of CPU time of the current process and thread (except sleep)
 - `gmtime([secs])`, `localtime([secs])`: Return the time as `time.struct_time` in [UTC](#) and your local [time zone](#)
 - `asctime([tm_struct])`, `ctime([secs])`: Return the time as a short string
 - `sleep(secs)`: Suspend execution of the current thread for the given seconds

- Usage example

```
import time
print(time.time())           # 1632946803.815631
print(time.process_time())   # 4.59375
print(time.thread_time())    # 3.890625
print(time.localtime())      # time.struct_time(..., tm_mday=30, tm_hour=5, ...)
print(time.gmtime())         # time.struct_time(..., tm_mday=29, tm_hour=20, ...)
print(time.ctime())          # Thu Sep 30 05:20:03 2021
```

```
start = time.time()
time.sleep(2)
elapsed = time.time() - start # 2.0132129000012355
print(elapsed)
```

glob: Unix-style **Pathname** Pattern Expansion

- glob finds all files and directories matched with a specified pattern used by the Unix shell.
 - **Wildcards** [\[Wikipedia\]](#)

Wildcard	Description	Example	Matches	Does not match
*	Everything including none	Law*	Law, Laws, Lawyer	GrokLaw, La, aw
		Law	Law, GrokLaw, Lawyer	La, aw
?	Any single character	?at	Cat, cat, Bat, bat	at
[abc]	One character given in the bracket	[CB]at	Cat, Bat	cat, bat, CBat
[a-z]	One character within the range	Letter[0-9]	Letter0, Letter1, ... , Letter9	Letters, Letter, Letter10

- API examples
 - `glob(pathname, *, recursive=False)`: Return a list of names of files and directories that match pathnames
- Usage example

```
# Please be aware where your working directory is (use 'pwd' in IPython console).
import glob                                # from glob import glob
glob.glob('*.py')                          # glob('*.py')
glob.glob('data/class_score_?.?.csv')      # glob('data/class_score_?.?.csv')
```


fnmatch: Unix-style **String** Pattern Matching

- fnmatch finds all **strings** matched with a specified pattern according to the rules used by the Unix shell.

- API examples

- `fnmatch(text, pattern)`: Test whether text matches the pattern (True or False; **case-insensitive**)
- `fnmatchcase(text, pattern)`: Similar to `fnmatch()` but **case-sensitive**
- `filter(iterable, pattern)`: Return a list of elements in iterable which satisfies pattern

- Usage example

```
import fnmatch
```

```
profs = [ 'My name is Choi and my E-mail is sunglok@seoultech.ac.kr.',  
          'My name is Kim and my e-mail address is jindae.kim@seoultech.ac.kr.' ]
```

```
# For a single string
```

```
print([fnmatch.fnmatch(prof, 'e-mail') for prof in profs])      # [False, False]  
print([fnmatch.fnmatch(prof, '*e-mail*') for prof in profs])    # [True, True]  
print([fnmatch.fnmatchcase(prof, '*e-mail*') for prof in profs]) # [False, True]  
print([fnmatch.fnmatchcase(prof, '*[Ee]-mail*') for prof in profs]) # [True, True]
```

```
# For a list of strings
```

```
print(fnmatch.filter(profs, '*e-mail*')) # ['My ... Choi ...', 'My ... Kim ...']  
print(fnmatch.filter(profs, '*Ch?i*'))   # ['My ... Choi ...']
```

csv: CSV File Reading and Writing

- csv contains classes to read and write tabular data in comma-separated values (CSV) format.
- API examples
 - `reader(file_obj, dialect='excel', **fmtparams)`: Return a reader object which can access lines in `file_obj`
 - `writer(file_obj, dialect='excel', **fmtparams)`: Return a writer object which can convert user data into comma-separated string on `file_obj`

- Usage example: Read all CSV files

Please be aware where your working directory is (use 'pwd' in IPython console).

```
import glob, csv
```

```
files = glob.glob('data/class_score_???.csv')
```

```
all_data = []
```

```
for file in files:
```

```
    with open(file, 'r') as f:      # Construct a file object
```

```
        csv_reader = csv.reader(f) # Construct a CSV reader object
```

```
        data = []
```

```
        for line in csv_reader:    # e.g. line = ['113', '86']
```

```
            if line and not line[0].strip().startswith('#'): # If 'line' is valid and not a header
```

```
                data.append([int(val) for val in line])      # Append 'line' to 'data' as numbers
```

```
        all_data = all_data + data                          # Merge 'data' to 'all_data'
```

[pickle](#): Python Object Serialization

- [pickle](#) provides binary protocols for [serializing](#) and de-serializing a Python object.
 - ~ You can save and load a Python object (in binary) without worry about its file format.
- API examples
 - `dump(obj, file_obj, ...)`: Write the given Python object `obj` into `file_obj`
 - `load(file_obj, ...)`: Read `file_obj` and return its Python object

- Usage example: Writing data to a file

```
# Please run the previous example, 'Read all CSV files'.
import pickle
with open('class_score_all.pickle', 'wb') as f:
    pickle.dump((files, all_data), f)
```

- Usage example: Loading data from the file

```
# Please be sure that 'class_score_all.pickle' was generated.
import pickle
with open('class_score_all.pickle', 'rb') as f:
    _, data = pickle.load(f)
    print(data)
```

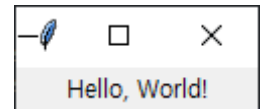
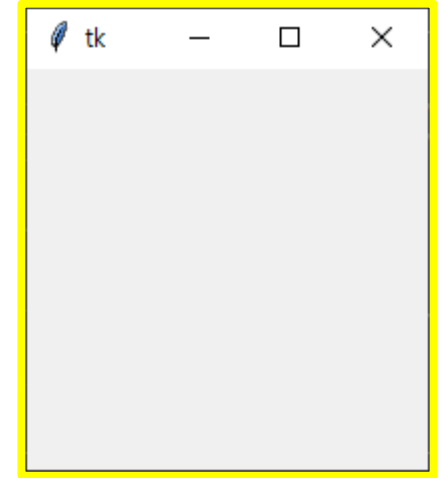
tkinter: Python Interface to Tcl/Tk GUI Toolkit

- tkinter is the standard Python interface to the Tcl/Tk GUI toolkit.
- API examples
 - David Amos, [Python GUI Programming with Tkinter](#), Real Python
 - 박응용, [tkinter – 편리한 GUI 툴킷](#), [파이썬 라이브러리](#), Wikidocs
- Usage example: Hello World

```
import tkinter as tk
```

```
root = tk.Tk()  
label = tk.Label(root, text='Hello, World!')  
label.pack()
```

```
root.mainloop()
```



tkinter: Python Interface to Tcl/Tk GUI Toolkit

- Usage example: A very simple chatbot

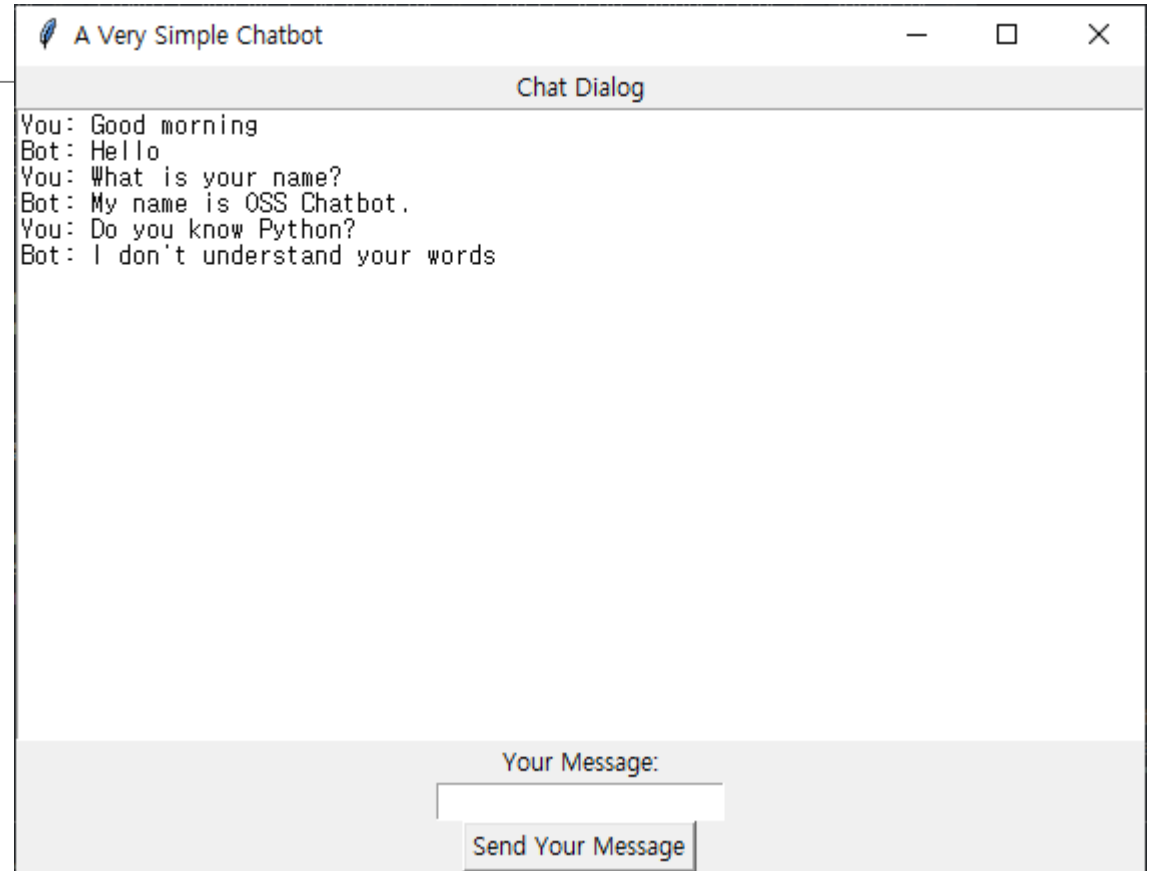
```
import tkinter as tk
from fnmatch import fnmatch

# Generate reply to the given message
def reply_msg(msg):
    if fnmatch(msg, '*hello*') or fnmatch(msg, '*good morning*'):
        return 'Hello'
    elif fnmatch(msg, '*what*you*name*'):
        return 'My name is OSS Chatbot.'
    return "I don't understand your words."

# Handle events from 'button_send'
def handle_button_send():
    text_dialog.insert('end', 'You: ' + entry_msg.get() + '\n')
    text_dialog.insert('end', 'Bot: ' + reply_msg(entry_msg.get()) + '\n')
    entry_msg.delete(0, tk.END) # Clear 'entry_msg' after reply

# Add widgets to GUI
root = tk.Tk()
root.title('A Very Simple Chatbot')
label = tk.Label(root, text='Chat Dialog')
label.pack()
text_dialog = tk.Text(root)
text_dialog.pack()
label = tk.Label(root, text='Your Message:')
label.pack()
entry_msg = tk.Entry(root)
entry_msg.pack()
button_send = tk.Button(root, text='Send Your Message', command=handle_button_send)
button_send.pack()

root.mainloop()
```



tkinter: Python Interface to Tcl/Tk GUI Toolkit

- Usage example: A very simple chatbot after [refactoring](#) (simple_chatbot.py)

```
import tkinter as tk
from fnmatch import fnmatch

class ChatBot:
    def __init__(self):
        self.talk_table = [
            ('*hello*', 'Hello'),
            ('*good morning*', 'Hello'),
            ('*what*you*name*', 'My name is OSS Chatbot.'),
        ]
        self.talk_unknown = "I don't understand your words."

    def reply(self, msg):
        for pattern, response in self.talk_table:
            if fnmatch(msg, pattern):
                return response
        return self.talk_unknown
```

2) Separate the data (talk_*) and algorithm (reply)

```
class SimpleChatBotGUI:
    def __init__(self, chatbot, master):
        self.chatbot = chatbot
        self.master = master
        self.master.title('A Very Simple Chatbot')
        self.label = tk.Label(master, text='Chat Dialog')
        self.label.pack()
        self.text_dialog = tk.Text(master)
        self.text_dialog.pack()
        self.label = tk.Label(master, text='Your Message:')
        self.label.pack()
        self.entry_msg = tk.Entry(master)
        self.entry_msg.pack()
        self.button_send = tk.Button(master, text='Send Your Message',
                                      command=self.handle_button)
        self.button_send.pack()

    def handle_button(self):
        msg = self.entry_msg.get()
        self.text_dialog.insert('end', 'You: ' + msg + '\n')
        self.text_dialog.insert('end', 'Bot: ' + self.chatbot.reply(msg) + '\n')
        self.entry_msg.delete(0, tk.END) # Clear 'entry_msg' after reply

if __name__ == '__main__':
    chatbot = ChatBot()
    root = tk.Tk()
    app = SimpleChatBotGUI(chatbot, root)
    root.mainloop()
```

1) Separate the model (ChatBot) and its view (SimpleChatBotGUI) [\[Wikipedia\]](#)

turtle: Turtle Graphics for Programming Education

- [Turtle graphics](#) was a vector drawing tool for introducing programming to kids.
 - It was a part of the original [Logo programming language](#), which has been re-implemented in Python with [tkinter](#).
- [API examples](#)
 - **Turtle** class
 - Motion: `forward()`, `backward()`, `right()`, `left()`, `goto()/setpos()`, `setx()`, `sety()`, `setheading()`, ...
 - Pen: `penup()`, `pendown()`, `pensize()`, `pencolor()`, ...
 - State: `pos()`, `heading()`, `distance()`, ... / `isdown()`, ...
 - **TurtleScreen/Screen** class
 - Event: `mainloop()`, `onkeypress()`, ...

- Usage example: Drawing a rectangle

If you use Spyder, please change your configuration as follows:

- Menu > Tools > Preference > **IPython console** > Graphics > **Graphics backend: Tkinter**

After your practice, please restore the configuration to 'Automatic' again.

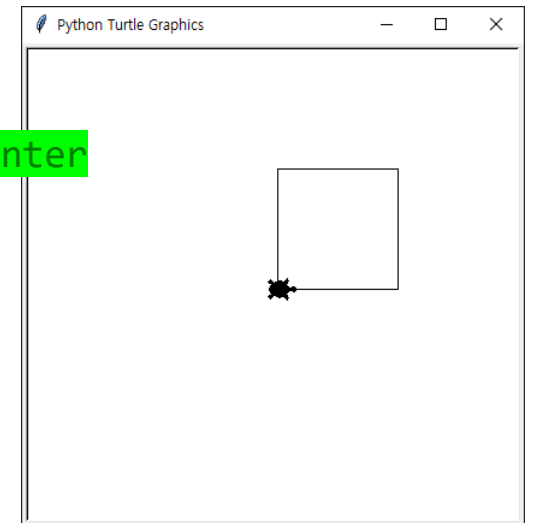
```
import turtle
```

```
turtle.shape('turtle')
```

```
for i in range(4):
```

```
    turtle.forward(100) # Unit: Pixel
```

```
    turtle.left(90)     # Unit: Degree
```



turtle: Turtle Graphics for Programming Education

- Usage example: Turtle painter (1/2)

```
import turtle

pen_pallate = ['black', 'red', 'green', 'blue', 'cyan', 'magenta', 'yellow']
pen_color = 0
pen_width = 3
pen_delta = 1
step_move = 10
step_turn = 10

def shift_pen_color():
    global pen_color
    pen_color = (pen_color + 1) % len(pen_pallate)
    turtle.pencolor(pen_pallate[pen_color])

def change_pen_width(delta):
    global pen_width
    pen_width += delta
    if pen_width < 1:
        pen_width = 1
    turtle.pensize(pen_width)

# Initialize the turtle
turtle.shape('turtle')
turtle.pencolor(pen_pallate[pen_color])
turtle.pensize(pen_width)
```

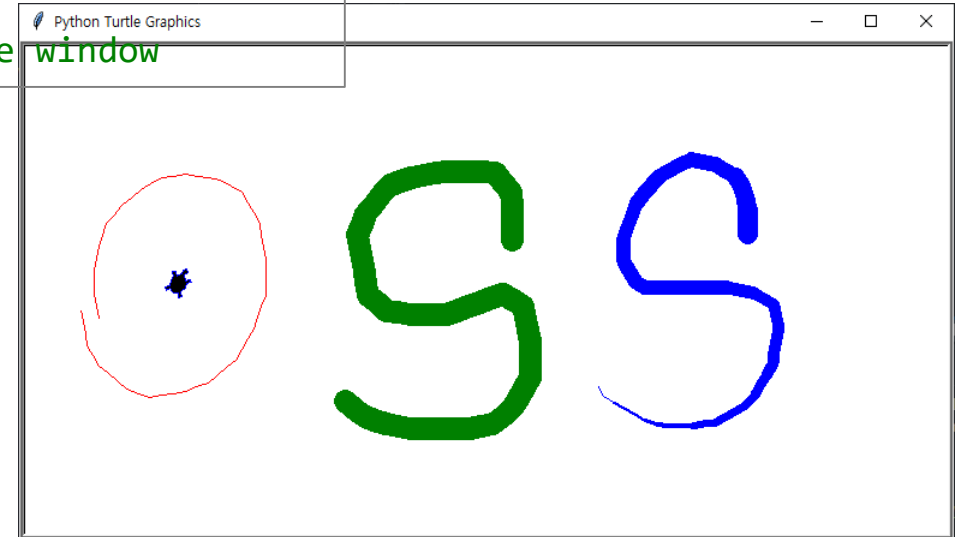


turtle: Turtle Graphics for Programming Education

- Usage example: Turtle painter (2/2)

```
# Register event handlers
turtle.onkeypress(shift_pen_color, 'c')
turtle.onkeypress(lambda: turtle.forward(step_move), 'Up')
turtle.onkeypress(lambda: turtle.backward(step_move), 'Down')
turtle.onkeypress(lambda: turtle.left(step_turn), 'Left')
turtle.onkeypress(lambda: turtle.right(step_turn), 'Right')
turtle.onkeypress(lambda: turtle.clear(), 'Escape')
turtle.onkeypress(lambda: turtle.pen(pendown=not turtle.isdown()), ' ')
turtle.onkeypress(lambda: change_pen_width(+pen_delta), 'm')
turtle.onkeypress(lambda: change_pen_width(-pen_delta), 'n')
turtle.listen()

#turtle.mainloop() # It is necessary out of Spyder, not to close the window
```



This example is not working in Spyder directly (F5 or Run)
Please type '!python turtlePainter.py' on IPython console in your Spyder.

```
import turtle
```

```
class TurtlePainter(turtle.RawTurtle):
```

```
    def __init__(self, canvas):  
        super().__init__(canvas)  
        self.pen_pallate = ['black', 'red', 'green', 'blue', 'cyan', 'magenta', 'yellow']  
        self.pen_color = 0  
        self.pen_width = 3  
        self.pen_delta = 1  
        self.step_move = 10  
        self.step_turn = 10
```

```
# Initialize the turtle
```

```
self.shape('turtle')  
self.pencolor(self.pen_pallate[self.pen_color])  
self.pensize(self.pen_width)
```

```
# Register event handlers
```

```
canvas.onkeypress(self.shift_pen_color, 'c')  
canvas.onkeypress(lambda: self.forward(self.step_move), 'Up')  
canvas.onkeypress(lambda: self.backward(self.step_move), 'Down')  
canvas.onkeypress(lambda: self.left(self.step_turn), 'Left')  
canvas.onkeypress(lambda: self.right(self.step_turn), 'Right')  
canvas.onkeypress(lambda: self.clear(), 'Escape')  
canvas.onkeypress(lambda: self.pen(pendown=not self.isdown()), ' ')  
canvas.onkeypress(lambda: self.change_pen_width(+self.pen_delta), 'm')  
canvas.onkeypress(lambda: self.change_pen_width(-self.pen_delta), 'n')  
canvas.listen()
```

```
def shift_pen_color(self):  
    self.pen_color = (self.pen_color + 1) % len(self.pen_pallate)  
    self.pencolor(self.pen_pallate[self.pen_color])
```

```
def change_pen_width(self, delta):  
    self.pen_width = max(self.pen_width + delta, 1)  
    self.pensize(self.pen_width)
```

- Usage example: Turtle painter after [refactoring](#)
(turtlePainter.py)

```
if __name__ == '__main__':  
    canvas = turtle.Screen()  
    painter = TurtlePainter(canvas)  
    # You can add another turtles by additional instantiation.  
    # another = turtle.Turtle('turtle')  
    # another.penup()  
    # another.goto(100, 100)  
    canvas.mainloop()
```



Beyond the [Python Standard Library](#)

- How to install a pre-built Python library (usually available in [PyPI](#))
 - Shell/Anaconda Prompt: `pip install package_name`
 - IPython console (in Spyder): `!pip install package_name`
- Example: Face detection using [OpenCV](#) (`face_detection.py`)
 - Install OpenCV: `pip install opencv-python`
 - Download a pre-trained model file, [haarcascade_frontalface_default.xml](#)
 - Prepare your test image or camera

Beyond the [Python Standard Library](#)

- Example: Face detection using [OpenCV](#) (face_detection.py)
 - Install OpenCV: `pip install opencv-python`
 - Download a pre-trained model file, [haarcascade_frontalface_default.xml](#)
 - Prepare your test image or camera

```
import cv2 as cv

# Load a face detector
face_detector = cv.CascadeClassifier('data/haarcascade_frontalface_default.xml')

# Prepare an image in gray scale
img = cv.imread('data/poster.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# Detect faces
faces = face_detector.detectMultiScale(gray)

# Visualize results
for (x, y, w, h) in faces:
    cv.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
cv.imshow('Face Detection using OpenCV', img)
cv.waitKey()
cv.destroyAllWindows()
```



tqdm: Progress Visualization in Console

- [tqdm](#) visualizes progress of iterations in console.
 - Note) The name, *tqdm*, came from *taqadum* (progress in Arabic).

- Simple visualization

```
n = 10000
for i in range(n):
    pass # Do something
    print(f'{i} / {n} ({100*i//n}%)') # Print progress
```

```
# ...
# 9998 / 10000 (100%)
# 9999 / 10000 (100%)
```

- Usage example

```
from tqdm import tqdm
n = 10000
for i in tqdm(range(n)):
    pass # Do something
```

```
# 76% | ██████████ 7568/10000 [00:33<00:10, 229.00it/s]
```

Summary

- [math](#): Mathematical Functions
- [decimal](#): Decimal Fixed-point and Floating-point Arithmetic
- [random](#): Pseudo-random Number Generators
- [time](#): Time Access and Conversions
- [glob](#): Unix-style Pathname Pattern Expansion
- [fnmatch](#): Unix-style String Pattern Matching
- [csv](#): CSV File Reading and Writing
- [pickle](#): Python Object Serialization
- [tkinter](#): Python Interface to Tcl/Tk GUI Toolkit
 - **Refactoring**
- [turtle](#): Turtle Graphics for Programming Education
 - **Refactoring**
- Beyond the [Python Standard Library](#)
 - `pip install package_name`
- [tqdm](#): Progress Visualization in Console