BIOS6642

Functions in Python Standard Library

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Functions and modules**
- Built-in functions
- Mathematical functions
- Other functions
- Turtle graphics

Functions and modules

- We have used some Python built-in functions in previous lectures, such as *type, int, float, str, print,* etc.

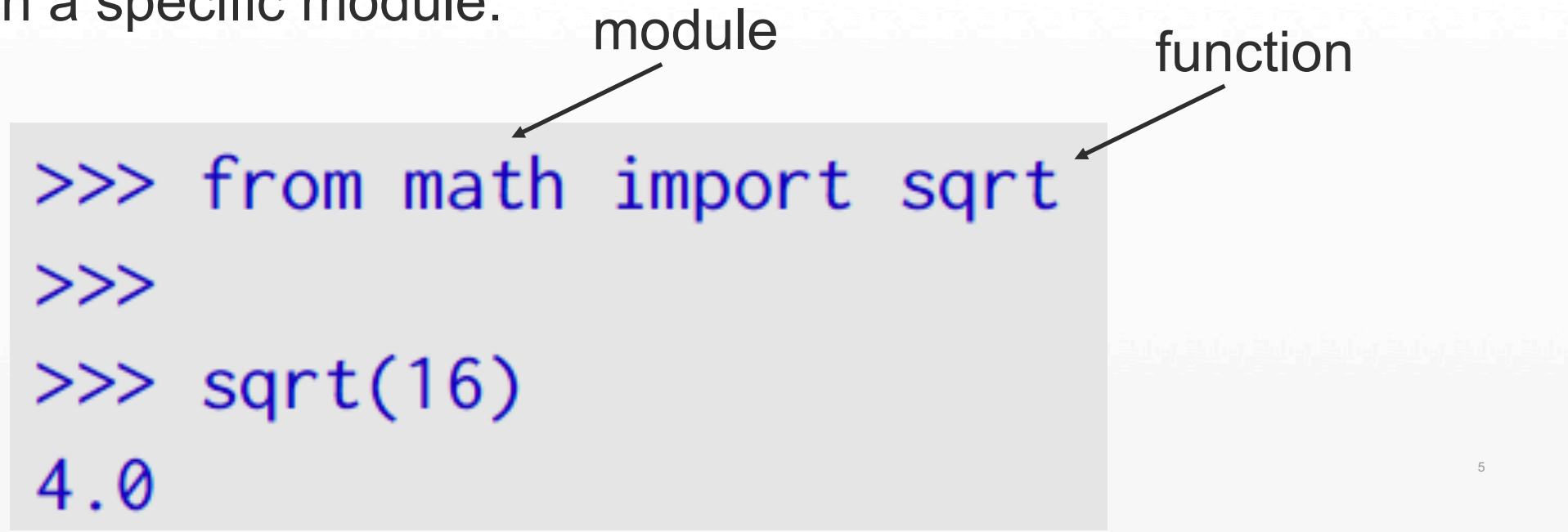
```
>>> type(42)
<class 'int'>
>>> int('32')
32
>>> int(3.99999)
3
>>> int(-2.3)
-2
>>> float(32)
32.0
>>> float('3.14159')
3.14159
```

Functions and modules

- For using functions in the Python standard library, programmers are concerned more about what the function does, not how it does it.
- Callers do not need to know the details of the code inside the function in order to use it, but want to know:
  - Function name
  - Function parameters
  - Result type

Functions and modules

- The Python standard library provides a set of modules that contain a lot of useful functions.
- A **module** is a file that contains a collection of related functions.
- The name of the file dictates the name of the module.
- Programmers need to use an *import* statement to get access to the functions in a specific module.



Functions and modules

- Import functions

- The function list is a comma-separated list of function names to import.

Functions and modules

- From the *math* module import the *sqrt* function

```
from math import sqrt
```

- From the *math* module import the *sqrt, log10*, and *cos* functions

```
from math import sqrt, log10, cos
```

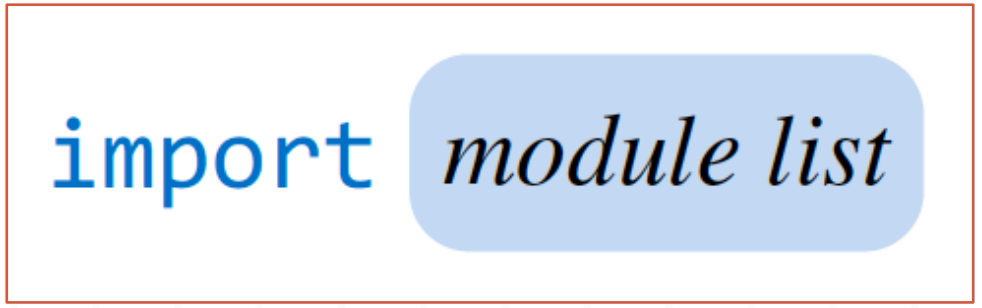
Functions and modules

- Example

```
from math import sqrt

#  Get value from the user
num = float(input("Enter number: "))

#  Compute the square root
root = sqrt(num)

#  Report result
print("Square root of", num, "=", root)
```

Functions and modules

- Import modules



- The module list is a comma-separated list of module names to import.
- This statement imports the entire module into the program.

Functions and modules

- Import modules

```
import math

y = math.sqrt(x)
print(math.log10(100))
```

- For large and complex programs, importing the entire module might be more compelling than importing the functions, because the complete name unambiguously identifies the function with its module.

Functions and modules

- Other methods to import functions from a module

- This statement import everything from the module into the program.

Functions and modules

- Other methods to import functions from a module

- This statement import everything from the module into the program.
- **This statement might "pollute" the program's namespace.**

Functions and modules

- The *dir()* function returns the list of names in the current local scope (i.e., namespace).

```
>>> from math import *
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__n
ame__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asin
h', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial
', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'in
f', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', '
log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radi
ans', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Functions and modules

- *help(exp)* shows:

```
Help on built-in function exp in module math:

exp(...)
    exp(x)

    Return e raised to the power of x.
```

Functions and modules

- After we reassign *exp=None,* the *help(exp)* shows:

```
Help on NoneType object:

class NoneType(object)
 |  Methods defined here:
 |
 |  __bool__(self, /)
 |      self != 0
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  __repr__(self, /)
 |      Return repr(self).
```

Functions and modules

- Import a module/function under a different name
  - Use only if necessary

```
import math as m
y = m.sqrt(x)
print(m.log10(100))
```

```
from math import sqrt as sq


print(sq(16))
```

Outline

- Functions and modules
- **Built-in functions**
- Mathematical functions
- Other functions
- Turtle graphics

Built-in functions

- Python standard library: https://docs.python.org/3/library/

Built-in functions

- The built-in functions, such as *int, float, str, print, input* and so on, in a module named *__builtins__* are automatically available to any Python program, i.e., no *import* statement is required.

```
>>> print('Hi')
Hi
>>> __builtins__.print('Hi')
Hi
>>> print
<built-in function print>
>>> __builtins__.print
<built-in function print>
>>> id(print)
9506056
>>> id(__builtins__.print)
9506056
```

id(x) gets the address in memory of object named *x*

Built-in functions

- Functions in the *__builtins__* module
- Link:

https://docs.python.org/3/library/functions.html

```
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'Bytes Warning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'Depre cationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingP ointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'Interrup tedError',
'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError',
'MemoryError', 'NameError', 'None', 'NotADirectoryEr ror',
'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'PermissionError', 'ProcessL ookupError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError',
'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEn codeError',
'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
'ValueError', 'Warning', 'WindowsError', 'Zero DivisionError',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__',
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii',
'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir',
'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float',
'format', 'frozenset', 'getattr ', 'globals', 'hasattr', 'hash', 'help',
'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit',
'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type',
'vars', 'zip']
```

Built-in functions

- Functions in the *__builtins__* module

```
'__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii',
'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir',
'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float',
'format', 'frozenset', 'getattr ', 'globals', 'hasattr', 'hash', 'help',
'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len',
'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next',
'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit',
'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice',
'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type',
'vars', 'zip']
```

Built-in functions

- The *help* function in the *__builtins__* module can print human-readable information about specific functions in the current namespace.

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Outline

- Functions and modules
- Built-in functions
- **Mathematical functions**
- Other functions
- Turtle graphics

Mathematical functions

A few functions in the *math* module

math Module	
`sqrt`	Computes the square root of a number: `sqrt(x)` = $\sqrt{x}$
`exp`	Computes e raised a power: `exp(x)` = e^x
`log`	Computes the natural logarithm of a number: `log(x)` = $\log_e x = \ln x$
`log10`	Computes the common logarithm of a number: `log(x)` = $\log_{10} x$
`cos`	Computes the cosine of a value specified in radians: `cos(x)` = $\cos x$; other trigonometric functions include sine, tangent, arc cosine, arc sine, arc tangent, hyperbolic cosine, hyperbolic sine, and hyperbolic tangent
`pow`	Raises one number to a power of another: `pow(x,y)` = x^y
`fabs`	Computes the absolute value of a number: `fabs(x)` = $\lvert x \rvert$

Mathematical functions

- Functions in the *math* module

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'g amma', 'hypot',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
```

Mathematical functions

Print prime numbers up to a value given by the user

```
from math import sqrt

max_value = int(input('Display primes up to what value? '))
value = 2  # Smallest prime number

while value <= max_value:
    # See if value is prime
    is_prime = True  # Provisionally, value is prime
    # Try all possible factors from 2 to value - 1
    trial_factor = 2
    root = sqrt(value)  # Compute the square root of value
    while trial_factor <= root:
        if value % trial_factor == 0:
            is_prime = False    # Found a factor
            break               # No need to continue; it is NOT prime
        trial_factor += 1       # Try the next potential factor
    if is_prime:
        print(value, end= ' ')  # Display the prime number
    value += 1                  # Try the next potential prime number

print()  # Move cursor down to next line
```

Outline

- Functions and modules
- Built-in functions
- Mathematical functions
- **Other functions**
- Turtle graphics

Other functions

- The *time* module contains a number of functions that relate to time.
- The *time.clock* function allows us to measure the time of parts of a program's execution.

Codes

```
from time import clock

sum = 0             # Initialize sum accumulator
start = clock()  # Start the stopwatch
for n in range(1, 100000001):    #  Sum the numbers
    sum += n
elapsed = clock() - start  #  Stop the stopwatch
print("sum:", sum, "time:", elapsed)  # Report results
```

Output

```
sum: 5000000050000000 time: 24.922694830903826
```

Other functions

- The *time.sleep* function suspends the program's execution for a specified number of seconds.

```
from time import sleep

for count in range(10, -1, -1):  # Range 10, 9, 8, ..., 0
    print(count)         # Display the count
    sleep(1)             # Suspend execution for 1 second
```

- The *time.sleep* function is useful for controlling the speed of graphical animations.

Other functions

- The Python *random* module contains a number of standard functions that can be used to generate pseudorandom numbers.
- A few of the functions from the *random* module:

randomfunctions Module	
`random`	Returns a pseudorandom floating-point number x in the range $0 \leq x < 1$
`randrange`	Returns a pseudorandom integer value within a specified range.
`seed`	Sets the random number seed.
`choice`	Selects an element at random from a collection of elements.

Other functions

- The *random.seed* function establishes the initial value from which the sequence of pseudorandom numbers is generated.

```
from random import randrange, seed

seed(23)                                    # Set random number seed
for i in range(0, 100):                     # Print 100 random numbers
    print(randrange(1, 1001), end=' ')      # Range 1...1,000, inclusive
print()                                     # Print newine
```

Other functions

- The *random.choice* function selects a random element from a collection of elements.

Codes

```
from random import choice

for i in range(10):
    print(choice(("one", "two", "three", "four", "five", "six",
                  "seven", "eight", "nine", "ten")))
```

Output

```
three
nine
five
nine
three
seven
nine
two
eight
ten
```

Other functions

- The *sys* module provides a number of functions and variables that give programmers access to system-specific information.

```
import sys

sum = 0
while True:
    x = int(input('Enter a number (999 ends):'))
    if x == 999:
        sys.exit(0)
    sum += x
    print('Sum is', sum)
```

- The *sys.exit* function accepts a single integer argument, which it passes back to the operating system when the program completes. The value zero indicates that the program completed successfully; a nonzero value represents abnormal termination.

Outline

- Functions and modules
- Built-in functions
- Mathematical functions
- Other functions
- **Turtle graphics**

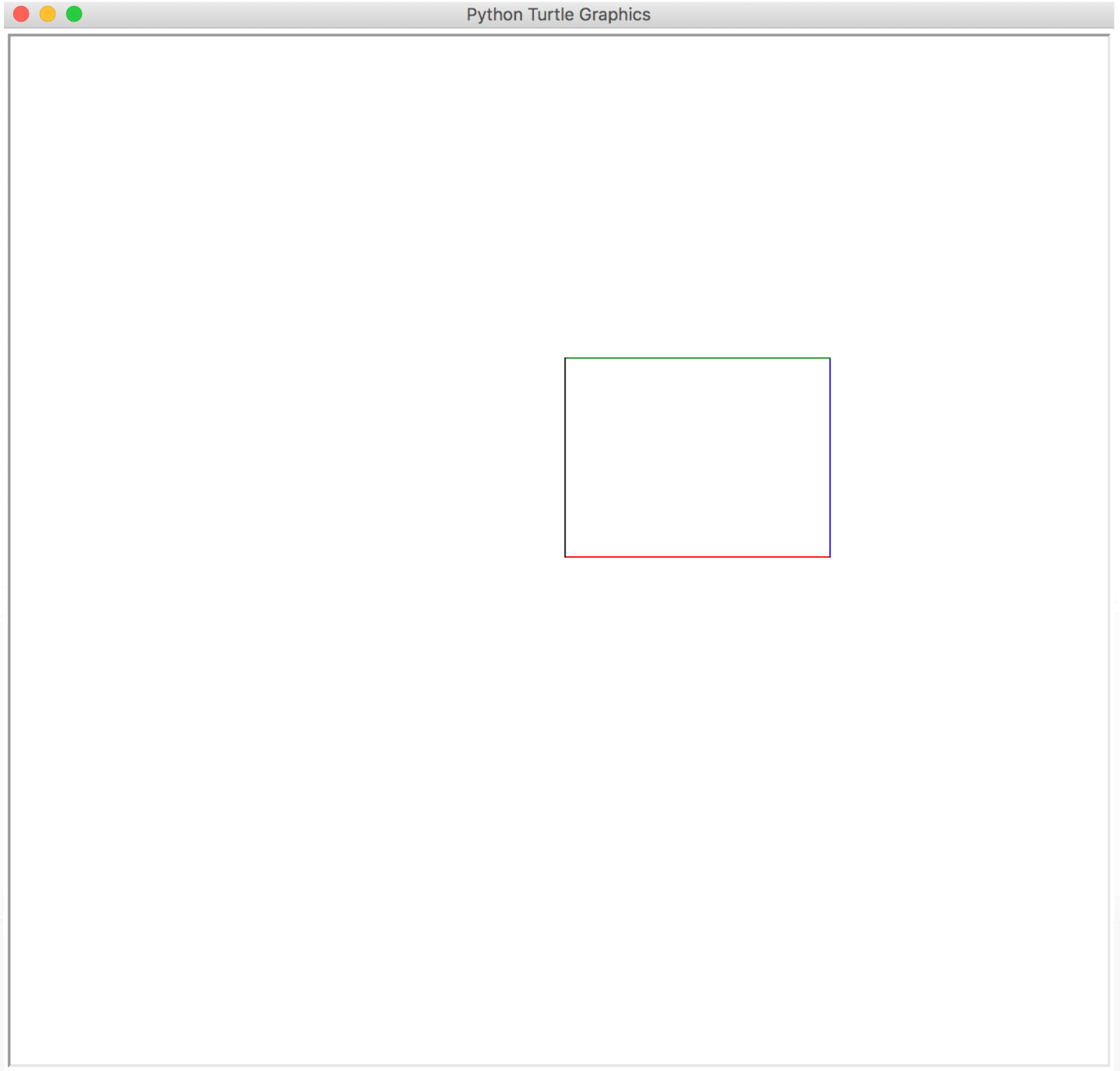
Turtle graphics

- The *turtle* module provides a set of functions to draw graphics.
- Example: draw a rectangle

Codes

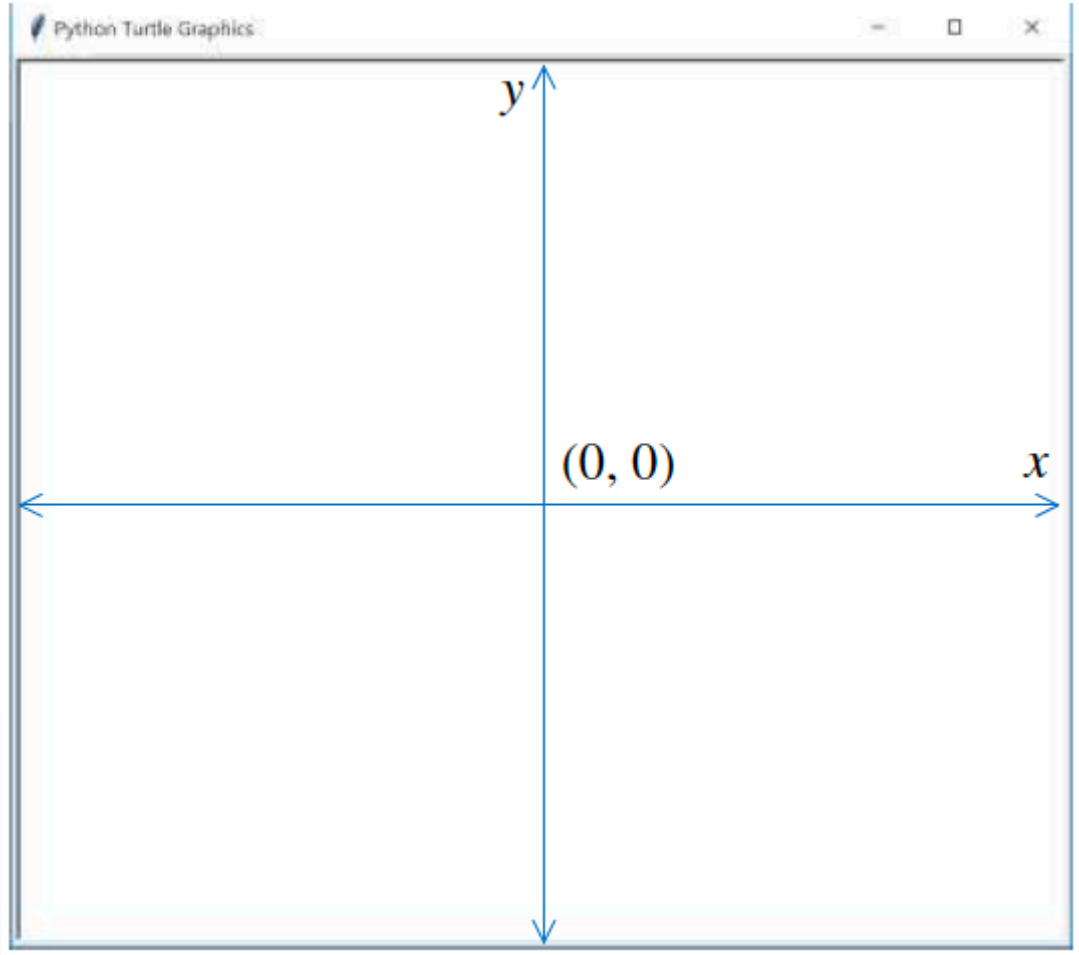
```
import turtle

turtle.pencolor('red')    # Set pen color to red
turtle.forward(200)       # Move pen forward 200 units (create bottom of rectangle)
turtle.left(90)           # Turn pen by 90 degrees
turtle.pencolor('blue')   # Change pen color to blue
turtle.forward(150)       # Move pen forward 150 units (create right wall)
turtle.left(90)           # Turn pen by 90 degrees
turtle.pencolor('green')  # Change pen color to green
turtle.forward(200)       # Move pen forward 200 units (create top)
turtle.left(90)           # Turn pen by 90 degrees
turtle.pencolor('black')  # Change pen color to black
turtle.forward(150)       # Move pen forward 150 units (create left wall)
turtle.hideturtle()       # Make pen invisible
turtle.exitonclick()      # Wait for user input
```

Output



Turtle graphics

- By default, the pen starts at the center of the window facing to the right. The default coordinate system is shown below.



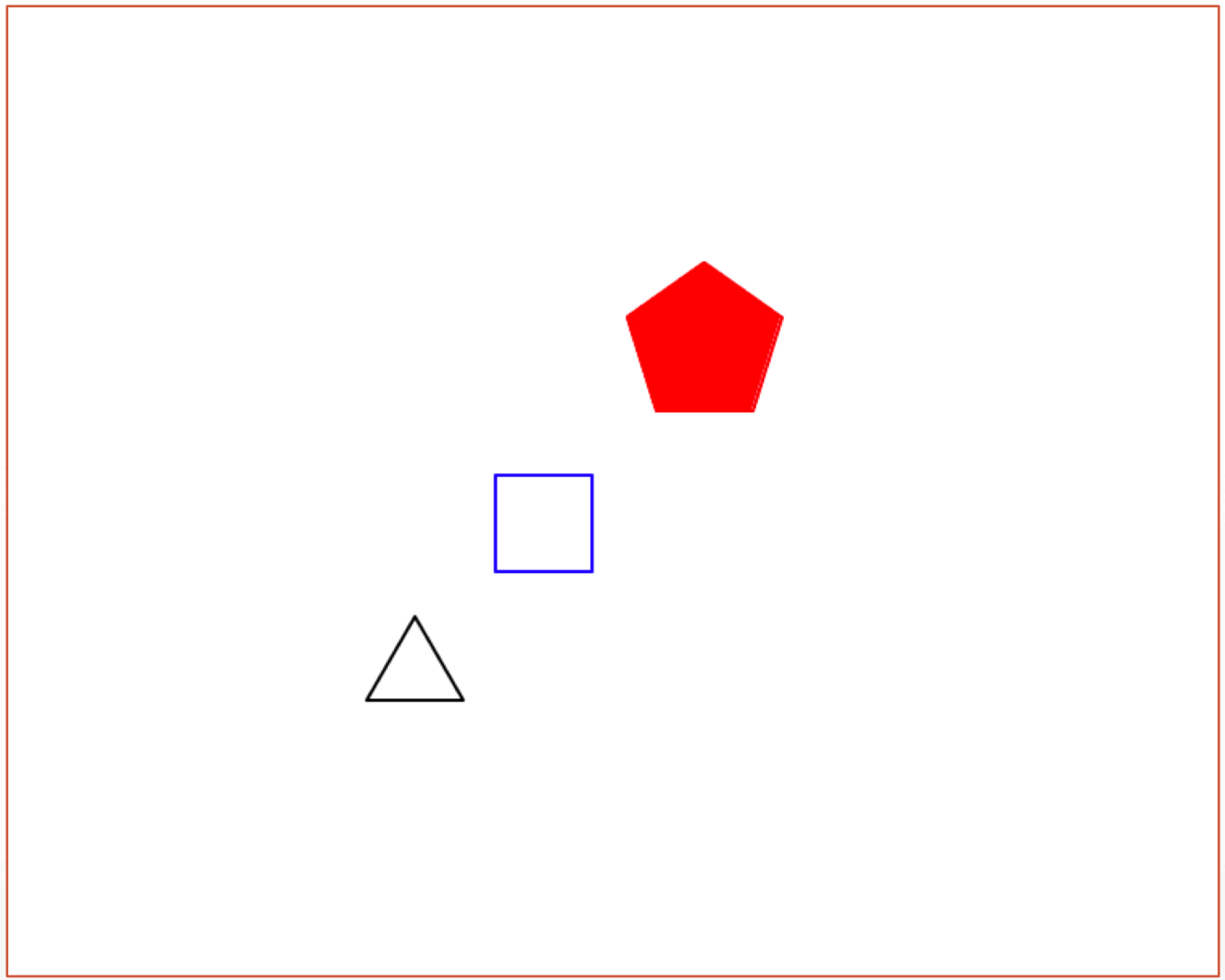
Turtle graphics

Draw a triangle, a square and a pentagon.

```
import turtle
import random


# Draws a regular polygon with the given number of sides.
# The length of each side is length.
# The pen begins at point(x, y).
# The color of the polygon is color (defaults to black).
# The polygon is rendered solid if fill is True (defaults to False).
def polygon(sides, length, x, y, color="black", fill=False):
    turtle.penup()
    turtle.setposition(x, y)
    turtle.pendown()
    turtle.color(color)
    if fill:
        turtle.begin_fill()
    for i in range(sides):
        turtle.forward(length)
        turtle.left(360//sides)
    if fill:
        turtle.end_fill()


# Disable rendering to speed up drawing
turtle.hideturtle()
turtle.tracer(0)

# Draw a few polygons
polygon(3, 30, 10, 10)                  # Back triangle outline
polygon(4, 30, 50, 50, "blue")          # Blue square outline
polygon(5, 30, 100, 100, "red", True)   # Red solid pentagon

turtle.update()  # Render image
turtle.exitonclick()  # Wait for user's mouse click
```

Turtle graphics

Output



Readings

- Think Python, chapters 3 and 4

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman