# Chapter 3
# Mathematical Functions, Strings, and Objects

## Motivations

Suppose you need to estimate the area enclosed by four cities, given the GPS locations (latitude and longitude) of these cities, as shown in the following diagram. How would you write a program to solve this problem? You will be able to write such a program after completing this chapter.

## Objectives

- To solve mathematics problems by using the functions in the **math** module (§3.2).
- To represent and process strings and characters (§§3.3-3.4).
- To encode characters using ASCII and Unicode (§§3.3.1-3.3.2).
- To use the **ord** to obtain a numerical code for a character and **chr** to convert a numerical code to a character (§3.3.3).
- To represent special characters using the escape sequence (§3.3.4).
- To invoke the **print** function with the end argument (§3.3.5).
- To convert numbers to a string using the **str** function (§3.3.6).
- To use the + operator to concatenate strings (§3.3.7).
- To read strings from the console (§3.3.8).
- To introduce objects and methods (§3.5).
- To format numbers and strings using the **format** function (§3.6).
- To draw various shapes (§3.7).
- To draw graphics with colors and fonts (§3.8).

# Built-in Functions and math Module

```
>>> max(2, 3, 4) # Returns a maximum number
4
>>> min(2, 3, 4) # Returns a minimu number
2
>>> round(3.51) # Rounds to its nearest integer
4
>>> round(3.4) # Rounds to its nearest integer
3
>>> abs(-3) # Returns the absolute value
3
>>> pow(2, 3) # Same as 2 ** 3
8
```

# The math Functions

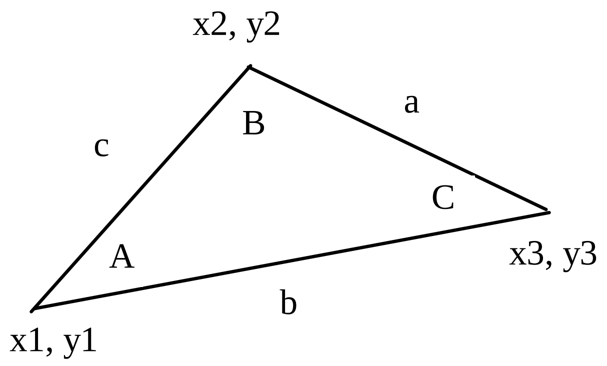
| Function | Description | Example |
|---|---|---|
| fabs(x) | Returns the absolute value of the argument. | fabs(-2) is 2 |
| ceil(x) | Rounds x up to its nearest integer and returns this integer. | ceil(2.1) is 3<br>ceil(-2.1) is -2 |
| floor(x) | Rounds x down to its nearest integer and returns this integer. | floor(2.1) is 2<br>floor(-2.1) is -3 |
| exp(x) | Returns the exponential function of x (e^x). | exp(1) is 2.71828 |
| log(x) | Returns the natural logarithm of x. | log(2.71828) is 1.0 |
| log(x, base) | Returns the logarithm of x for the specified base. | log10(10, 10) is 1 |
| sqrt(x) | Returns the square root of x. | sqrt(4.0) is 2 |
| sin(x) | Returns the sine of x. x represents an angle in radians. | sin(3.14159 / 2) is 1<br>sin(3.14159) is 0 |
| asin(x) | Returns the angle in radians for the inverse of sine. | asin(1.0) is 1.57<br>asin(0.5) is 0.523599 |
| cos(x) | Returns the cosine of x. x represents an angle in radians. | cos(3.14159 / 2) is 0<br>cos(3.14159) is -1 |
| acos(x) | Returns the angle in radians for the inverse of cosine. | acos(1.0) is 0<br>acos(0.5) is 1.0472 |
| tan(x) | Returns the tangent of x. x represents an angle in radians. | tan(3.14159 / 4) is 1<br>tan(0.0) is 0 |
| fmod(x, y) | Returns the remainder of x/y as double. | fmod(2.4, 1.3) is 1.1 |
| degrees(x) | Converts angle x from radians to degrees | degrees(1.57) is 90 |
| radians(x) | Converts angle x from degrees to radians | radians(90) is 1.57 |

MathFunctions

Run

## Problem: Compute Angles

Given three points of a triangle, you can compute the angles using the following formula:



```
A = acos((a * a - b * b - c * c) / (-2 * b * c))
B = acos((b * b - a * a - c * c) / (-2 * a * c))
C = acos((c * c - b * b - a * a) / (-2 * a * b))
```

ComputeAngles

Run

## Strings and Characters

A string is a sequence of characters. *String* literals can be enclosed in matching *single quotes* (') or *double quotes* ("). Python does not have a data type for characters. A single-character string represents a character.

```
letter = 'A' # Same as letter = "A"
numChar = '4' # Same as numChar = "4"
message = "Good morning"
# Same as message = 'Good morning'
```

## NOTE

-For consistency, this book uses double quotes for a string with more than one character and single quotes for a string with a single character or an empty string.

-This convention is consistent with other programming languages. So, it will be easy to convert a Python program to a program written in other languages.

# Unicode and ASCII Code

Python characters use *Unicode*, a 16-bit encoding scheme Python supports Unicode. Unicode is an encoding scheme for representing international characters. ASCII is a small subset of Unicode.

DisplayUnicode

Run

# Appendix B: ASCII Character Set

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

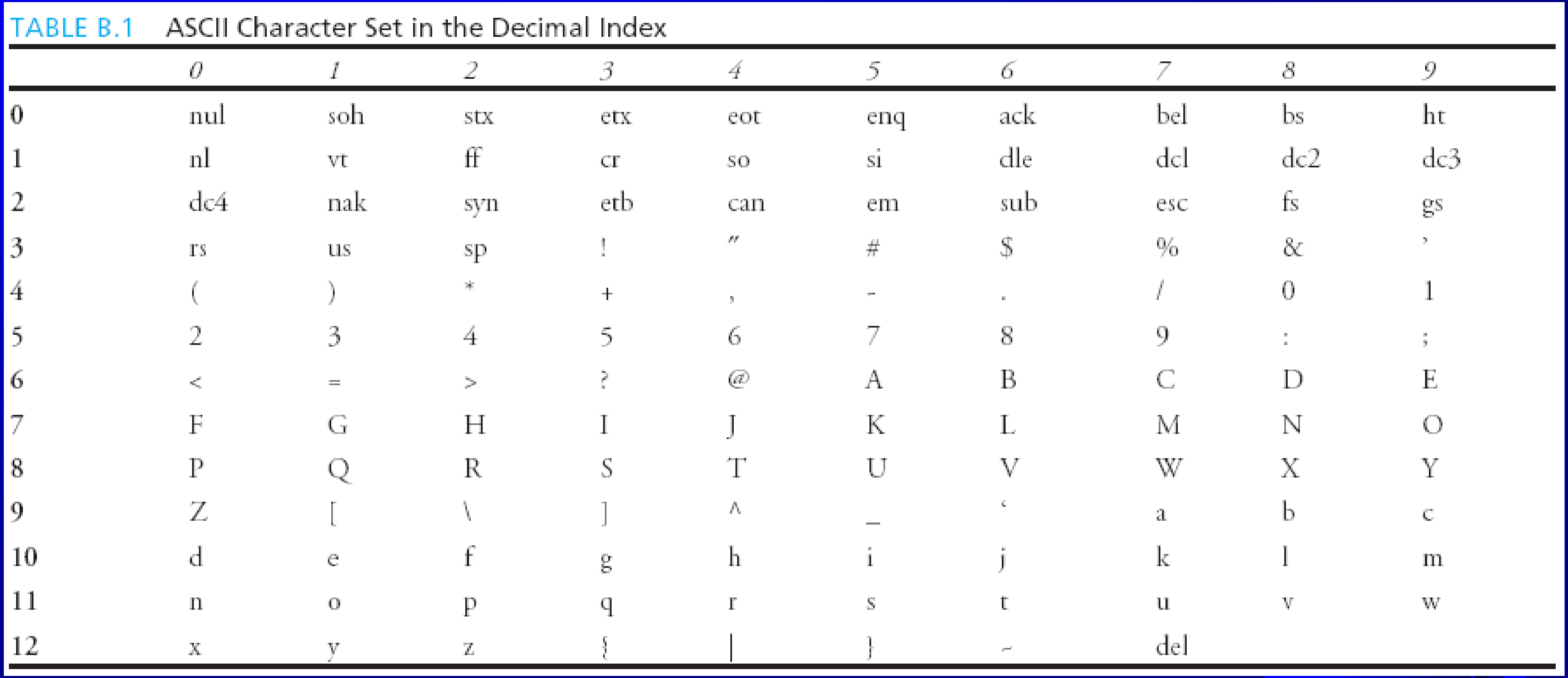


TABLE B.1 ASCII Character Set in the Decimal Index

| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | soh | stx | etx | eot | enq | ack | bel | bs | ht |
| 1 | nl | vt | ff | cr | so | si | dle | dcl | dc2 | dc3 |
| 2 | dc4 | nak | syn | etb | can | em | sub | esc | fs | gs |
| 3 | rs | us | sp | ! | " | # | $ | % | & | ' |
| 4 | ( | ) | * | + | , | - | . | / | 0 | 1 |
| 5 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; |
| 6 | < | = | > | ? | @ | A | B | C | D | E |
| 7 | F | G | H | I | J | K | L | M | N | O |
| 8 | P | Q | R | S | T | U | V | W | X | Y |
| 9 | Z | [ | \ | ] | ^ | _ | ` | a | b | c |
| 10 | d | e | f | g | h | i | j | k | l | m |
| 11 | n | o | p | q | r | s | t | u | v | w |
| 12 | x | y | z | { | \| | } | ~ | del | | |

# ASCII Character Set, cont.

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

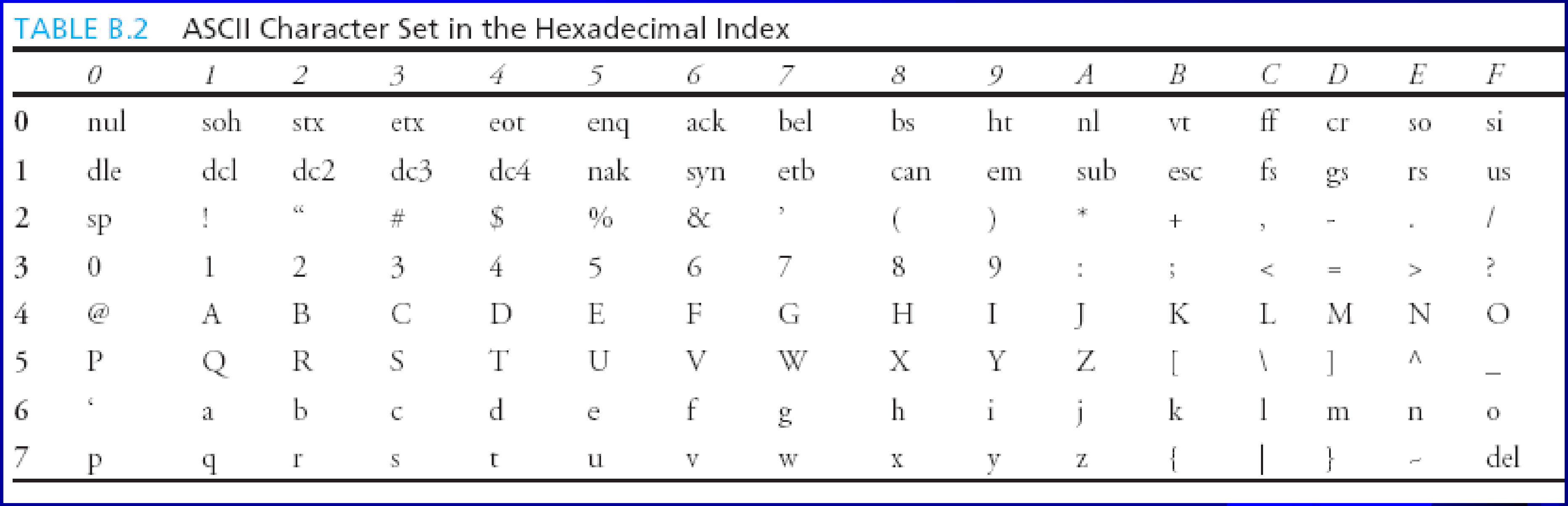


TABLE B.2 ASCII Character Set in the Hexadecimal Index

| | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *A* | *B* | *C* | *D* | *E* | *F* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | nul | soh | stx | etx | eot | enq | ack | bel | bs | ht | nl | vt | ff | cr | so | si |
| 1 | dle | dcl | dc2 | dc3 | dc4 | nak | syn | etb | can | em | sub | esc | fs | gs | rs | us |
| 2 | sp | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 6 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | del |

# Functions ord and chr

```
>>> ch = 'a'
>>> ord(ch)
>>> 97
>>> chr(98)
>>> 'b'
```

Python **ord()** function takes string argument of a single Unicode character and return its integer Unicode code point value.

Python **chr()** function takes integer argument and return the string representing a character at that code point.

## Escape Sequences for Special Characters

| *Description* | *Escape Sequence* | *Unicode* |
|---|---|---|
| Backspace | `\b` | `\u0008` |
| Tab | `\t` | `\u0009` |
| Linefeed | `\n` | `\u000A` |
| Carriage return | `\r` | `\u000D` |
| Backslash | `\\` | `\u005C` |
| Single Quote | `\'` | `\u0027` |
| Double Quote | `\"` | `\u0022` |

## Printing without the Newline

```
print(item, end = 'anyendingstring')
```

```
print("AAA", end = ' ')
print("BBB", end = '')
print("CCC", end = '***')
print("DDD", end = '***')
```

- You can use the optional named argument **end** to explicitly mention the string that should be appended at the end of the line.
- Whatever you provide as the **end** argument is going to be the terminating string.
- So if you provide an empty string, then no newline characters, and no spaces will be appended to your input.

```
# use the named argument "end" to explicitly specify the end of line string
print("Hello World!", end = ‘’)
print("My name is Selim")
# output:
# Hello World!My name is Selim
```

## The str Function

The str function can be used to convert a number into a string. For example,

```
>>> s = str(3.4) # Convert a float to string
>>> s
'3.4'
>>> s = str(3) # Convert an integer to string
>>> s
'3'
>>>
```

# The String Concatenation Operator

You can use the + operator add two numbers. The + operator can also be used to concatenate (combine) two strings. Here are some examples:

```
>>> message = "Welcome " + "to " + "Python"
>>> message
'Weclome to Python'
>>> chapterNo = 2
>>> s = "Chapter " + str(chapterNo)
>>> s
'Chapter 2'
>>>
```

# Reading Strings from the Console

To read a string from the console, use the input function. For example, the following code reads three strings from the keyboard:

```
s1 = input("Enter a string: ")
s2 = input("Enter a string: ")
s3 = input("Enter a string: ")
print("s1 is " + s1)
print("s2 is " + s2)
print("s3 is " + s3)
```

## Case Study: Minimum Number of Coins

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

ComputeChange

Run

## Introduction to Objects and Methods

In Python, all data—including numbers and strings—are actually objects.

An object is an entity. Each object has an id and a type. Objects of the same kind have the same type. You can use the **id** function and **type** function to get these information for an object.

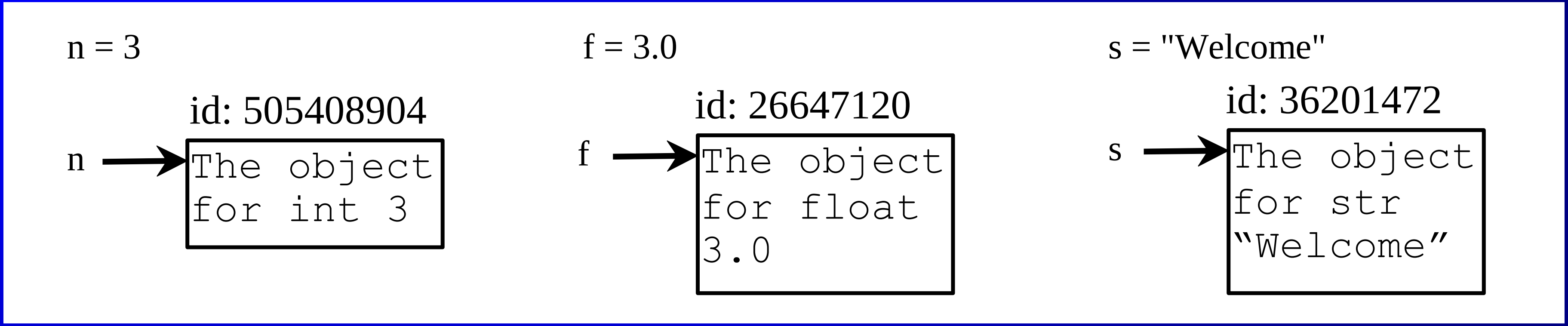
## Object Types and Ids

The **id** and **type** functions are rarely used in programming, but they are good pedagogical tools for understanding objects.

```
>>> n = 3  # n is an integer
>>> id(n)
505408904
>>> type(n)
<class ’int’>
>>> f = 3.0  # f is a float
>>> id(f)
26647120
>>> type(f)
<class ’float’>
```

```
>>> s = "Welcome" # s is a string
>>> id(s)
36201472
>>> type(s)
<class ’str’>
```

# OOP and str Objects



The **id** and **type** functions are rarely used in programming, but they are good pedagogical tools for understanding objects.

## Object vs. Object reference Variable

- For n = 3, we say n is an integer variable that holds value 3.
- Strictly speaking, n is a variable that references an int object for value 3.
- For simplicity, it is fine to say n is an int variable with value 3.

# Methods

- You can perform operations on an object. The operations are defined using functions.
- The functions for the objects are called *methods* in Python. Methods can only be invoked from a specific object.
- For example, the string type has the methods such as lower() and upper(), which returns a new string in lowercase and uppercase.
- Here are the examples to invoke these methods:

# str Object Methods

```
>>> s = "Welcome"
>>> s1 = s.lower() # Invoke the lower method
>>> s1
'welcome'
>>> s2 = s.upper() # Invoke the upper method
>>> s2
'WELCOME'
```

# Striping beginning and ending Whitespace Characters

Another useful string method is strip(), which can be used to strip the whitespace characters from the both ends of a string.

```
>>> s = "\t Welcome \n"
>>> s1 = s.strip() # Invoke the strip method
>>> s1
'Welcome'
```

## Formatting Numbers and Strings

Often it is desirable to display numbers in certain format. For example, the following code computes the interest, given the amount and the annual interest rate.

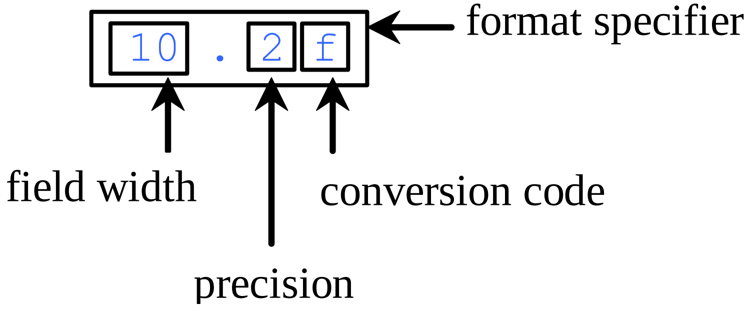
-The built-in format() function returns a formatted representation of the given value controlled by the format specifier.

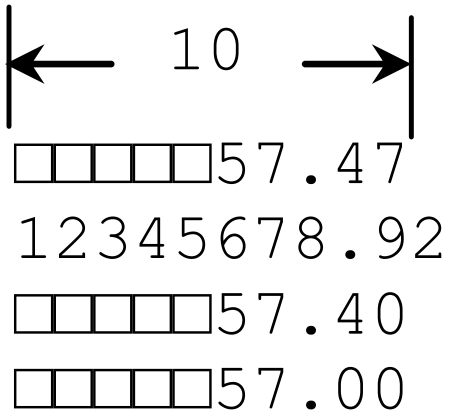
-The format function formats a number or a string and returns a string.

format(item, format-specifier)

# Formatting Floating-Point Numbers

```
print(format(57.467657, '10.2f'))
print(format(12345678.923, '10.2f'))
print(format(57.4, '10.2f'))
print(format(57, '10.2f'))
```



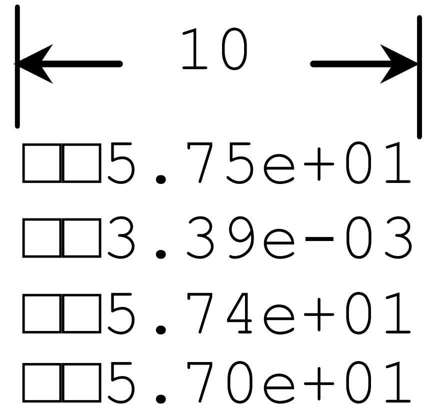


```
□□□□□57.47
12345678.92
□□□□□57.40
□□□□□57.00
```

# Formatting in Scientific Notation

If you change the conversion code from f to e, the number will be formatted in scientific notation. For example,

```
print(format(57.467657, '10.2e'))
print(format(0.0033923, '10.2e'))
print(format(57.4, '10.2e'))
print(format(57, '10.2e'))
```

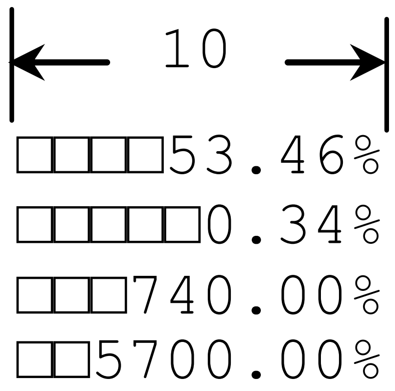


```
□□5.75e+01
□□3.39e-03
□□5.74e+01
□□5.70e+01
```

# Formatting as a Percentage

You can use the conversion code % to format numbers as a percentage. For example,

```
print(format(0.53457, '10.2%'))
print(format(0.0033923, '10.2%'))
print(format(7.4, '10.2%'))
print(format(57, '10.2%'))
```

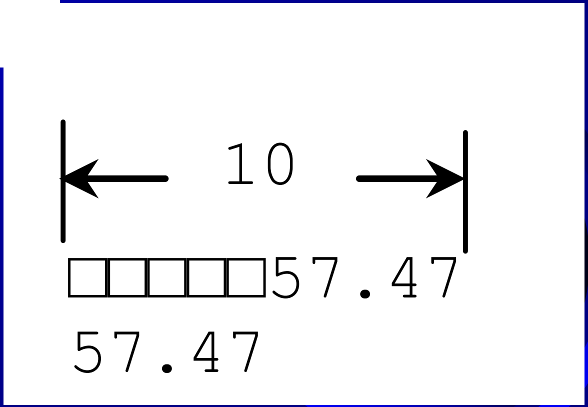


```
□□□□53.46%
□□□□□0.34%
□□□740.00%
□□5700.00%
```

# Justifying Format

By default, the format is right justified. You can put the symbol < in the format specifier to specify that the item is a left justified in the resulting format within the specified width. For example,

```
print(format(57.467657, '10.2f'))
print(format(57.467657, '<10.2f'))
```

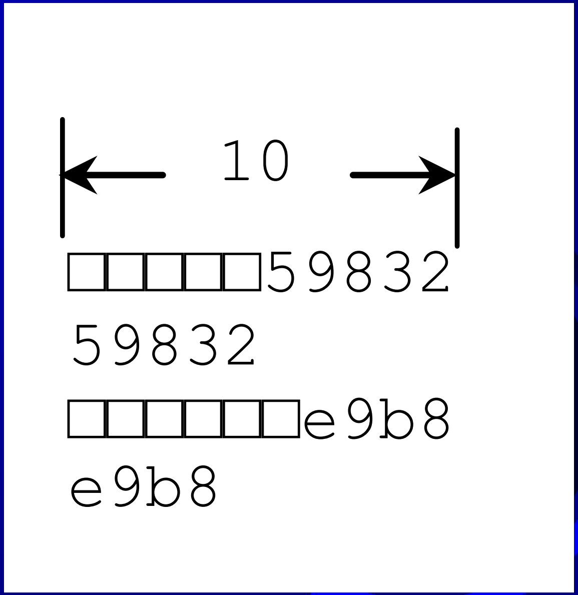


```
□□□□□57.47
57.47
```

# Formatting Integers

You can use the conversion code d, x, o, and b to format an integer in decimal, hexadecimal, octal, or binary. You can specify a width for the conversion. For example,

```
print(format(59832, '10d'))
print(format(59832, '<10d'))
print(format(59832, '10x'))
print(format(59832, '<10x'))
```

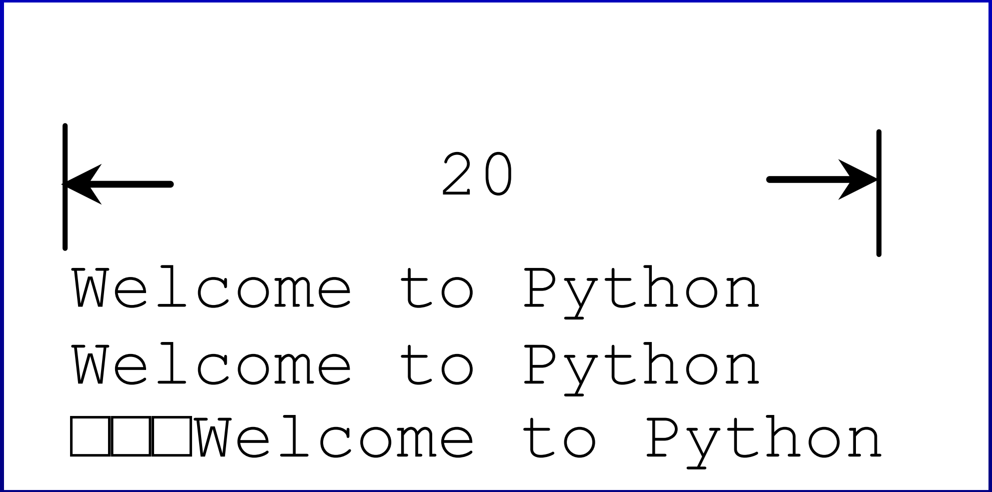


```
□□□□□59832
59832
□□□□□□e9b8
e9b8
```

# Formatting Strings

You can use the conversion code s to format a string with a specified width. For example,

```
print(format("Welcome to Python", '20s'))
print(format("Welcome to Python", '<20s'))
print(format("Welcome to Python", '>20s'))
```



```
Welcome to Python
Welcome to Python
□□□Welcome to Python
```

## Three Methods for String Formatting

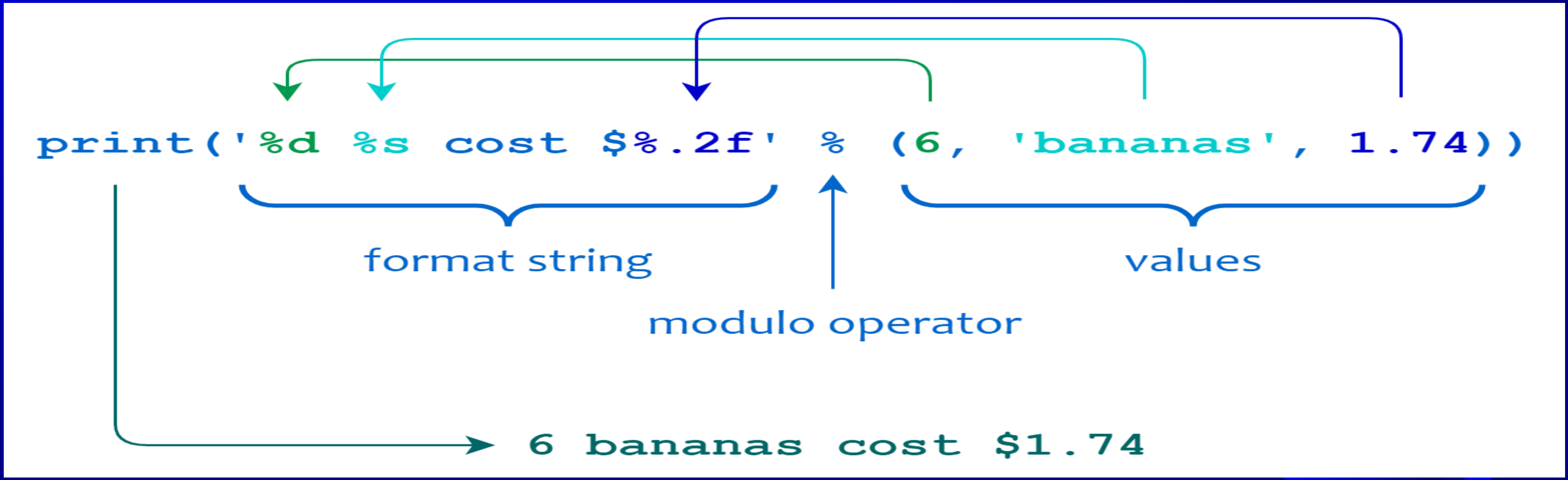
- String formatting is the process of infusing things in the string dynamically and presenting the string.
- Three String Formating Methods:
  - 1- Formatting with % Operator.
  - 2- Formatting with format() string method.
  - 3- Formatting with string literals, called f-strings.

# 1 – Modulo (%) String Formatting in Python

- <format_string> % <values>

```
> print("%d %s cost $%.2f" % (6, "bananas", 1.74))
```

```
6 bananas cost $1.74
```



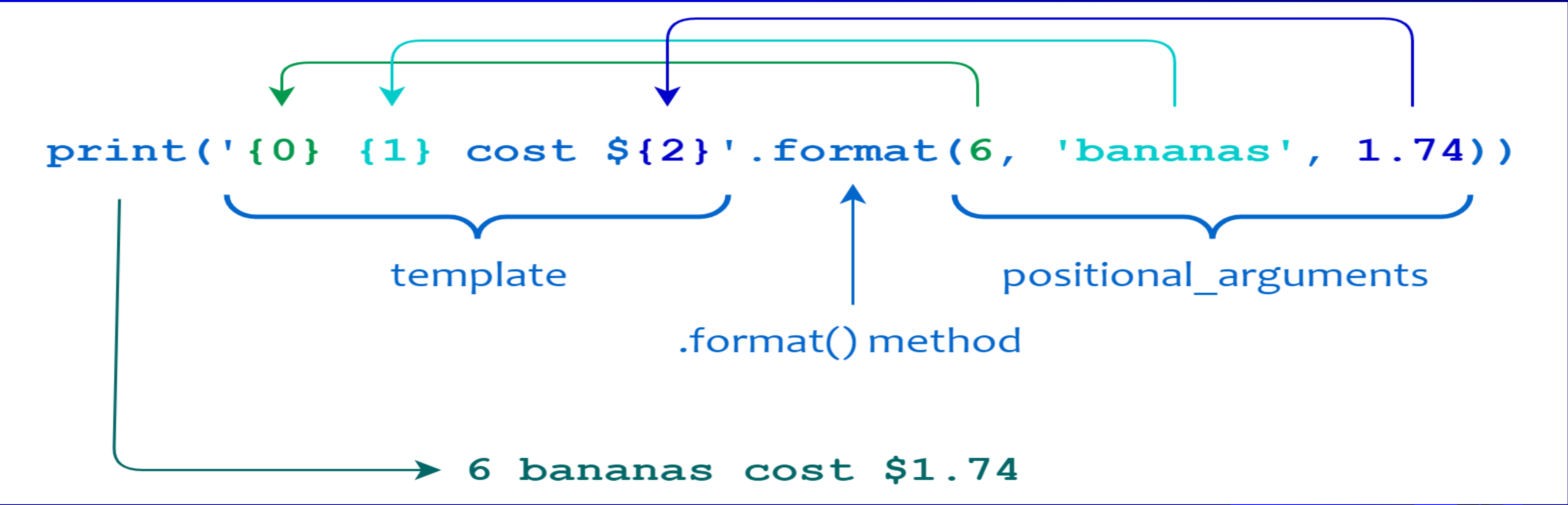
## 2-The String .format() Method

```
>print('The valueof pi is: {0:1.5f}'.format(3.141592))
        The value of pi is: 3.14159
```

- Syntax: {[index]:[width][.precision][type]}
- Type can be
  - ‘d’ for integers
  - ‘f’ for floating-point numbers
  - ‘b’ for binary numbers
  - ‘o’ for octal numbers
  - ‘x’ for octal hexadecimal numbers
  - ‘s’ for string
  - ‘e’ for floating-point in an exponent format

# 2-The String .format() Method with positional arguments

- print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))
- The replacement fields are {0}, {1}, and {2}

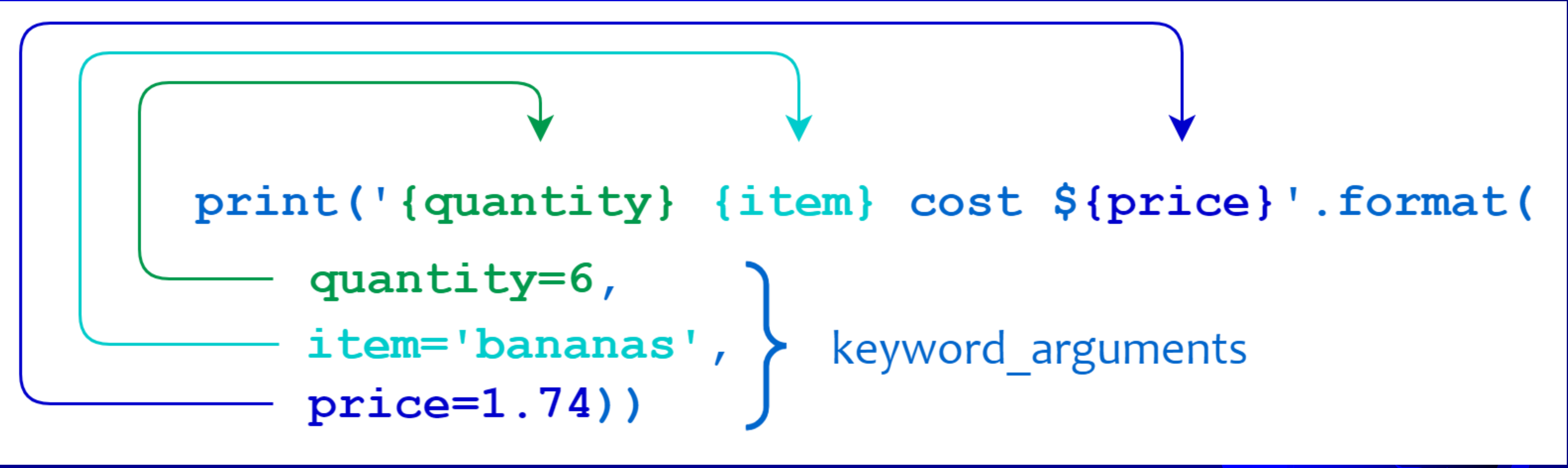


```
print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))
```

```
6 bananas cost $1.74
```

# 2-The String .format() Method with keyword arguments

- print('{quantity} {item} cost ${price}'.format(quantity=6, item='bananas', price=1.74))
- The replacement fields are {quantity}, {item}, and {price}.



```
print('{quantity} {item} cost ${price}'.format(
      quantity=6,
      item='bananas',
      price=1.74))
```

# 3- Formatting with string literals, called f-strings.

- To form an f-string, prefix the string with the letter “ f ”. The string itself can be formatted in much the same way that you would with str.format().
- F-strings provide a concise and convenient way to embed python expressions inside string literals for formatting.

```
>>> quantity = 6
>>> item = 'bananas’
>>> price = 1.74
>>> print(f'{quantity}  {item} cost ${price}’)
        6  bananas cost $1.74
```

## 3- Formatting with string literals, called f-strings.

- F-strings with format specifiers

Ex 1:

```
val = 12.3
print(f'{val:.2f}’)
print(f'{val:.5f}’)
          12.30
          12.30000
```

Ex 2:

```
a = 300
print(f"{a:x}") # hexadecimal
print(f"{a:o}") # octal
print(f"{a:e}") # scientific
    12c
    454
    3.000000e+02
```

# 3- Formatting with string literals, called f-strings.

- f-strings are faster and better than both %-formatting and str.format().
- f-strings expressions are evaluated are at runtime, and we can also embed expressions inside f-string, using a very simple and easy syntax.
- The expressions inside the braces are evaluated in runtime and then put together with the string part of the f-string and then the final string is returned.

```
>>> a = 5
>>> b = 10
>>> print(f"He said his age is {2 * (a + b)}.")
```

# Drawing Various Shapes

- turtle is a pre-installed Python library that enables users to create pictures and shapes by providing them with a virtual canvas.
- The onscreen pen that you use for drawing is called the turtle and this is what gives the library its name.
- In short, the Python turtle library helps new programmers get a feel for what programming with Python is like in a fun and interactive way.

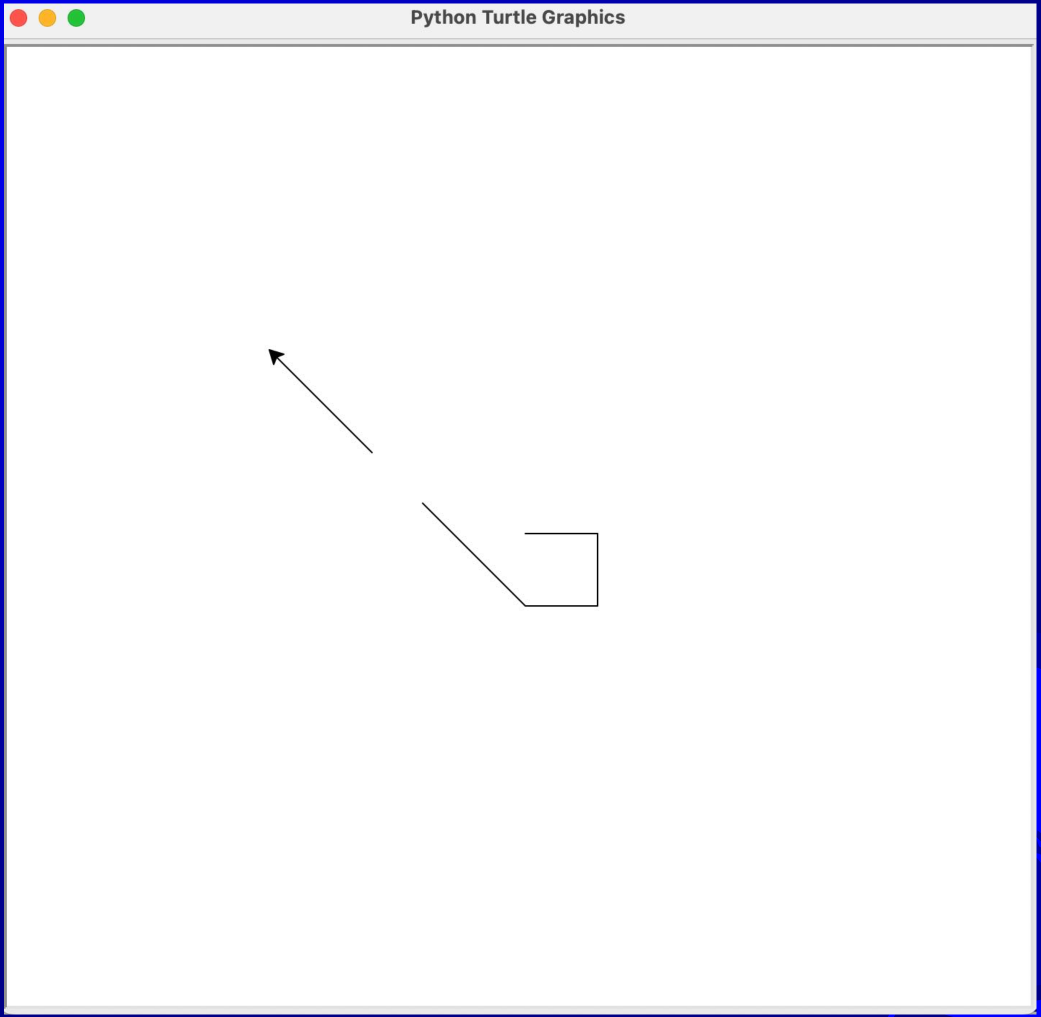
- A turtle contains methods for moving the pen and setting the pen’s size and speed.

# Turtle Pen Drawing State Methods

| Method | Description |
|---|---|
| turtle.pendown() | Pull the pen down - drawing when moving. |
| turtle.penup() | Pull the pen up – no drawing when moving. |
| turtle.pensize(width) | Set the line thickness to width. |

# Example

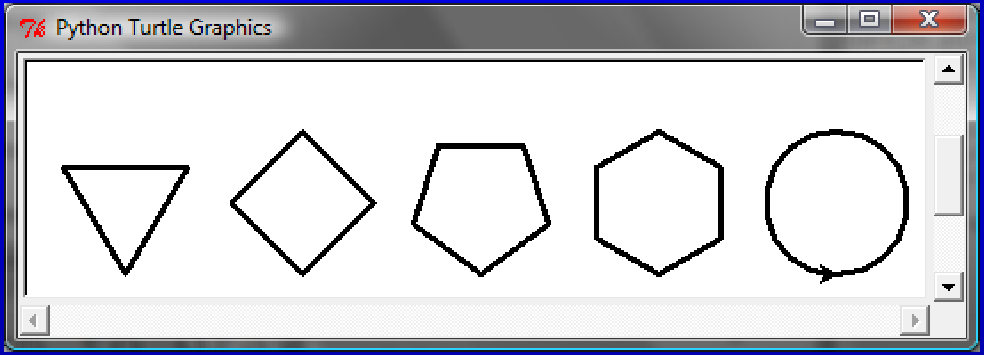
```
import turtle
t = turtle.Turtle()

t.forward(50)
t.right(90)
t.forward(50)
t.right(90)
t.forward(50)

t.right(45)
t.forward(100)
t.penup()
t.forward(50)
t.pendown()
t.forward(100)

turtle.done()
```



# Turtle Motion Methods

| Method | Description |
|---|---|
| `turtle.forward(d)` | Move the turtle forward by the specified distance in the direction the turtle is headed. |
| `turtle.backward(d)` | Move the turtle backward by the specified distance in the opposite direction the turtle is headed. Do not change the turtle's direction. |
| `turtle.right(angle)` | Turn turtle right by the specified angle. |
| `turtle.left(angle)` | Turn turtle left by the specified angle. |
| `turtle.goto(x, y)` | Move turtle to an absolute position. |
| `turtle.setx(x)` | Move turtle's x-coordinate to a specified position. |
| `turtle.sety(y)` | Move turtle's y-coordinate to a specified position. |
| `turtle.setheading(angle)` | Set the orientation of the turtle to a specified angle. 0-East, 90-North, 180-West, 270-South. |
| `turtle.home()` | Move turtle to the origin to (0, 0) and east direction. |
| `turtle.circle(r, ext, step)` | Draw a circle with the specified radius, extent, and step. |
| `turtle.dot(d, color)` | Draw a circle dot with the specified diameter and color. |
| `turtle.undo()` | Undo (repeatedly) the last turtle action(s). |
| `turtle.speed(s)` | Set turtle's speed to an integer between 0 and 10. |

# Problem: Draw Simple Shapes



SimpleShapes

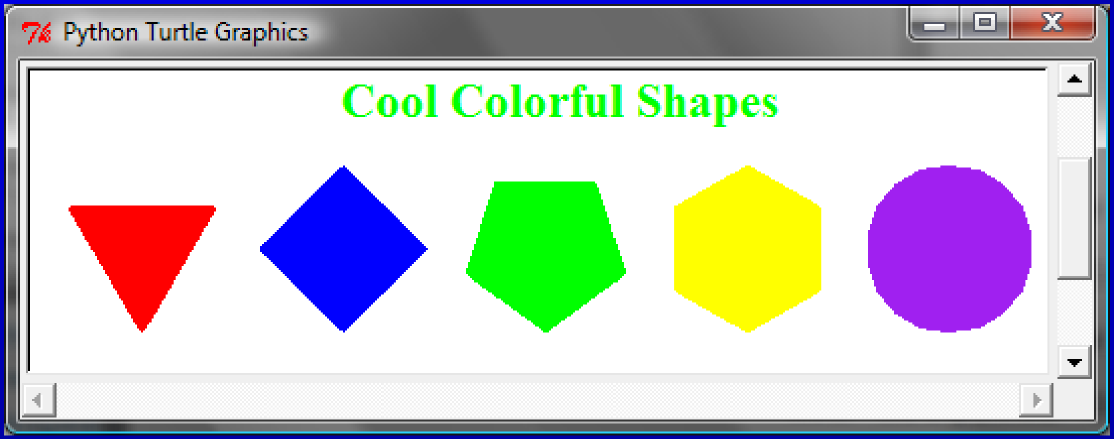
Run

## Turtle Drawing with Colors and Fonts

| **Method** | **Description** |
|---|---|
| `turtle.color(c)` | Set the pen color. |
| `turtle.fillcolor(c)` | Set the pen fill color. |
| `turtle.begin_fill()` | Call this method before filling a shape. |
| `turtle.end_fill()` | Fill the shapes drawn before the last call to begin_fill. |
| `turtle.isFilling()` | Return the fill state. True if filling. |
| `turtle.clear()` | Clear the window. State and the position of the turtle are not effected. |
| `turtle.reset()` | Clear the window and reset the state and position to the original default value. |
| `turtle.screensize(w, h)` | Set the width and height of the cancas. |
| `turtle.hideturtle()` | Make the turtle invisible. |
| `turtle.hideturtle()` | Make the turtle visible. |
| `turtle.isvisible()` | Return True if the turtle is visible. |
| `turtle.write(s, font=("Arial", 8, "normal"))` | Write the string s on the turtle position with the optional font. Font is a triple consisting of fontname, fontsize, and fonttype. |

# Drawing with Colors and Fonts

A turtle object contains the methods for setting colors and fonts.



ColorShapes

Run