

Arithmetic Expressions

Table 6 Arithmetic Expression Examples

Mathematical Expression	Python Expression	Comments
$\frac{x + y}{2}$	<code>(x + y) / 2</code>	The parentheses are required; <code>x + y / 2</code> computes $x + \frac{y}{2}$.
$\frac{xy}{2}$	<code>x * y / 2</code>	Parentheses are not required; operators with the same precedence are evaluated left to right.
$\left(1 + \frac{r}{100}\right)^n$	<code>(1 + r / 100) ** n</code>	The parentheses are required.
$\sqrt{a^2 + b^2}$	<code>sqrt(a ** 2 + b ** 2)</code>	You must import the <code>sqrt</code> function from the <code>math</code> module.
π	<code>pi</code>	<code>pi</code> is a constant declared in the <code>math</code> module.

Roundoff Errors

- Floating point values are not exact
 - This is a limitation of binary values; not all floating point numbers have an exact representation

- Open PyCharm, open a new file and type in:

```
price = 4.35
quantity = 100
total = price * quantity
# Should be 100 * 4.35 = 435.00
print(total)
```

- You can deal with roundoff errors by
 - rounding to the nearest integer (see Section [2.2.4](#))
 - or by displaying a fixed number of digits after the decimal separator (see Section [2.5.3](#)).

Unbalanced Parentheses

- Consider the expression
 $((a + b) * t / 2 * (1 - t))$
 - What is wrong with the expression?
- Now consider this expression.
 $(a + b) * t) / (2 * (1 - t))$
 - This expression has three “(“ and three “)”, but it still is not correct
- At any point in an expression the count of “(“ must be greater than or equal to the count of “)”
- At the end of the expression the two counts must be the same

Additional Programming Tips

- Use Spaces in expressions

```
totalCans = fullCans + emptyCans
```

- Is easier to read than

```
totalCans=fullCans+emptyCans
```

- Other ways to import modules:

```
From math import sqrt, sin, cos # imports the functions listed
```

```
From math import * # imports all functions from the module
```

```
Import math # imports all functions from the module
```

- If you use the last style you have to add the module name and a “.” before each function call

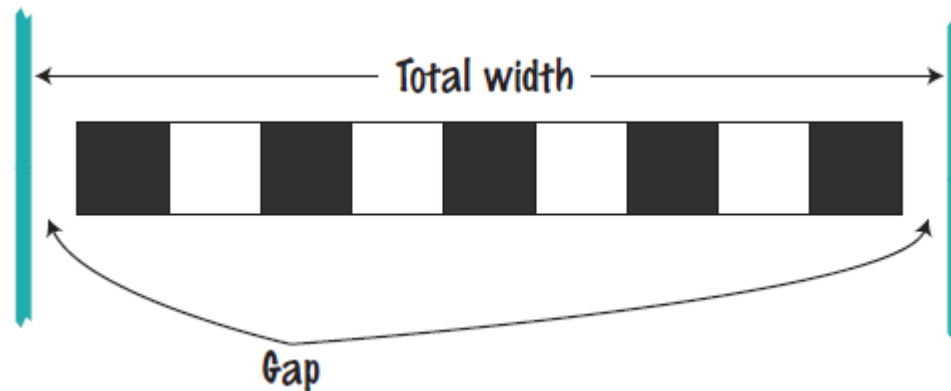
```
y = math.sqrt(x)
```

2.3 Problem Solving

DEVELOP THE ALGORITHM FIRST, THEN
WRITE THE PYTHON

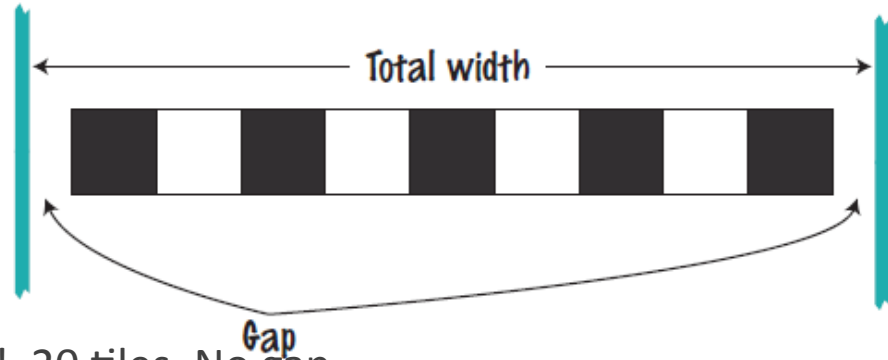
2.3 Problem Solving: First by Hand

- A very important step for developing an algorithm is to first carry out the computations by hand.
 - If you can't compute a solution by hand, how do you write the program?
- Example Problem:
 - A row of black and white tiles needs to be placed along a wall. For aesthetic reasons, the architect has specified that the first and last tile shall be black.
 - Your task is to compute the number of tiles needed and the gap at each end, given the space available and the width of each tile.



Start with example values

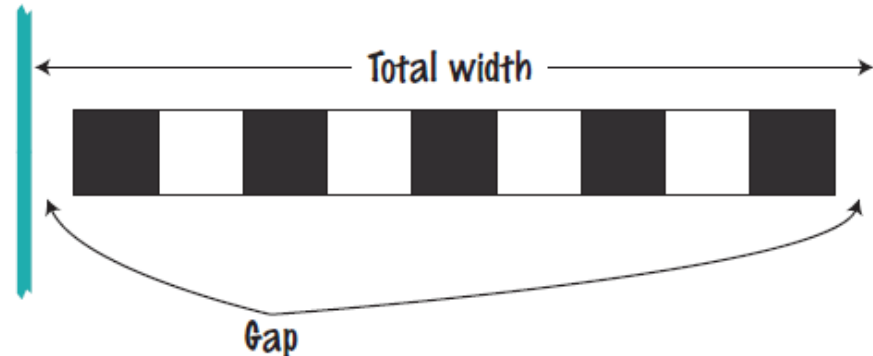
- Givens
- Total width: 100 inches
- Tile width: 5 inches
- Test your values
 - Let's see... $100/5 = 20$, perfect! 20 tiles. No gap.
 - But wait... BW...BW "...first and last tile shall be black."
- Look more carefully at the problem....
 - Start with one black, then some number of WB pairs



- Observation: each pair is 2x width of 1 tile
 - In our example, $2 \times 5 = 10$ inches

Keep applying your solution

- Total width: 100 inches
- Tile width: 5 inches



- Calculate total width of all tiles
 - One black tile: 5"
 - 9 pairs of BWs: 90"
 - Total tile width: 95"
- Calculate gaps (one on each end)
 - $100 - 95 = 5''$ total gap
 - $5'' \text{ gap} / 2 = 2.5''$ at each end

Now devise an algorithm

- Use your example to see how you calculated values
- How many pairs?
 - Note: must be a whole number
 - Integer part of: $(\text{total width} - \text{tile width}) / 2 \times \text{tile width}$
- How many tiles?
 - $1 + 2 \times$ the number of pairs
- Gap at each end
 - $(\text{total width} - \text{number of tiles} \times \text{tile width}) / 2$

The algorithm

- Calculate the number of pairs of tiles
 - Number of pairs = integer part of $(\text{total width} - \text{tile width}) / (2 * \text{tile width})$
- Calculate the number of tiles
 - Number of tiles = $1 + (2 * \text{number of pairs})$
- Calculate the gap
 - Gap at each end = $(\text{total width} - \text{number of tiles} * \text{tile width}) / 2$
- Print the number of pairs of tiles
- Print the total number of tiles in the row
- Print the gap

2.4 Strings

Strings

- Start with some simple definitions:
 - Text consists of **characters**
 - **Characters** are letters, numbers, punctuation marks, spaces,
 - A **string** is a sequence of **characters**
- In Python, string literals are specified by enclosing a sequence of **characters** within a matching pair of either single or double quotes.

```
print("This is a string.", 'So is this.')
```

- By allowing both types of delimiters, Python makes it easy to include an apostrophe or quotation mark within a string.
 - `message = 'He said "Hello"'`
 - Remember to use matching pairs of quotes, single with single, double with double

String Length

- The number of characters in a string is called the length of the string. (For example, the length of "Harry" is 5).
- You can compute the length of a string using Python's `len()` function:
`length = len("World!") # length is 6`
- A string of length 0 is called the empty string. It contains no characters and is written as `""` or `"`.

String Concatenation (“+”)

- You can ‘add’ one String onto the end of another

```
firstName = "Harry"
```

```
lastName = "Morgan"
```

```
name = firstName + lastName # HarryMorgan
```

```
print("my name is:", name)
```

- You wanted a space in between the two names?

```
name = firstName + " " + lastName # Harry Morgan
```

Using “+” to concatenate strings is an example of a concept called operator overloading. The “+” operator performs different functions of variables of different types

String repetition (“*”)

- You can also produce a string that is the result of repeating a string multiple times.
- Suppose you need to print a dashed line.
- Instead of specifying a literal string with 50 dashes, you can use the * operator to create a string that is comprised of the string "-" repeated 50 times.

```
dashes = "-" * 50
```

- results in the string
- "-----"

The "*" operator is also overloaded.

Converting Numbers to Strings

- Use the `str()` function to convert between numbers and strings.

- Open PyCharm, then open a new file and type in:

```
balance = 888.88
dollars = 888
balanceAsString = str(balance)
dollarsAsString = str(dollars)
print(balanceAsString)
print(dollarsAsString)
```

- To turn a string containing a number into a numerical value, we use the `int()` and `float()` functions:

```
id = int("1729")
price = float("17.29")
print(id)
print(price)
```

- **This conversion is important when the strings come from user input.**

Strings and Characters

- **strings** are sequences of **characters**
 - Python uses **Unicode** characters
 - **Unicode** defines over 100,000 characters
 - **Unicode** was designed to be able to encode text in essentially all written languages
 - Characters are stored as integer values
 - See the ASCII subset on Unicode chart in Appendix A
 - For example, the letter 'H' has a value of 72

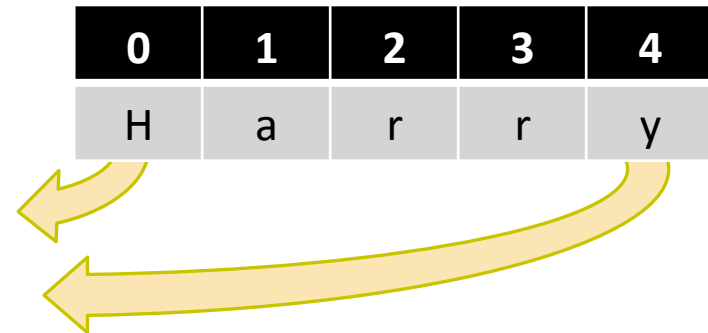
Copying a character from a String

- Each char inside a String has an index number:

0	1	2	3	4	5	6	7	8	9
c	h	a	r	s		h	e	r	e

- The first char is index zero (0)
- The [] operator returns a char at a given index inside a String:

```
name = "Harry"  
start = name[0]  
last = name[4]
```

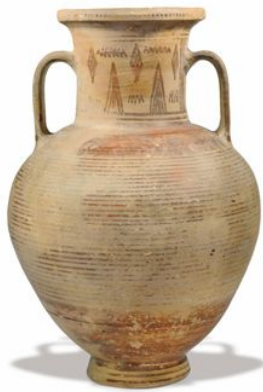


String Operations

Table 7 String Operations

Statement	Result	Comment
<pre>string = "Py" string = string + "thon"</pre>	string is set to "Python"	When applied to strings, + denotes concatenation.
<pre>print("Please" + " enter your name: ")</pre>	Prints Please enter your name:	Use concatenation to break up strings that don't fit into one line.
<pre>team = str(49) + "ers"</pre>	team is set to "49ers"	Because 49 is an integer, it must be converted to a string.
<pre>greeting = "H & S" n = len(greeting)</pre>	n is set to 5	Each space counts as one character.
<pre>string = "Sally" ch = string[1]</pre>	ch is set to "a"	Note that the initial position is 0.
<pre>last = string[len(string) - 1]</pre>	last is set to the string containing the last character in string	The last character has position len(string) - 1.

Data plus tools



Methods

- In computer programming, an object is a software entity that represents a value with certain behavior.
 - The value can be simple, such as a string, or complex, like a graphical window or data file.
- The behavior of an object is given through its **methods**.
 - A method is a collection of programming instructions to carry out a specific task – similar to a function
- But unlike a **function**, which is a standalone operation, a **method** can only be applied to an object of the type for which it was defined.
 - Methods are specific to a type of object
 - Functions are general and can accept arguments of different types
- You can apply the `upper()` method to any string, like this:
 - `name = "John Smith"`
 - `# Sets uppercaseName to "JOHN SMITH"`
 - `uppercaseName = name.upper()`

Some Useful String Methods

Table 8 Useful String Methods

Method	Returns
<code>s.lower()</code>	A lowercase version of string <i>s</i> .
<code>s.upper()</code>	An uppercase version of <i>s</i> .
<code>s.replace(<i>old</i>, <i>new</i>)</code>	A new version of string <i>s</i> in which every occurrence of the substring <i>old</i> is replaced by the string <i>new</i> .

String Escape Sequences

- How would you print a double quote?
 - Preface the " with a "\" inside the double quoted String

```
print("He said \"Hello\"")
```

- OK, then how do you print a backslash?
 - Preface the \ with another \

```
System.out.print("C:\\Temp\\Secret.txt")
```

- Special characters inside Strings
 - Output a newline with a '\n'

```
print("*\n**\n***\n")
```

```
*  
**  
***
```

2.5 Input and Output

Input and Output

- You can read a String from the console with the input() function:
 - `name = input("Please enter your name")`
- Converting a String variable to a number can be used if numeric (rather than string input) is needed
 - `age = int(input("Please enter age: "))`
 - The above is equivalent to doing it two steps (getting the input and then converting it to a number):
 - `aString = input("Please enter age: ")` # String input
 - `age = int(aString)` # Converted to
 - # int

Formatted output

- Outputting floating point values can look strange:

Price per liter: 1.21997

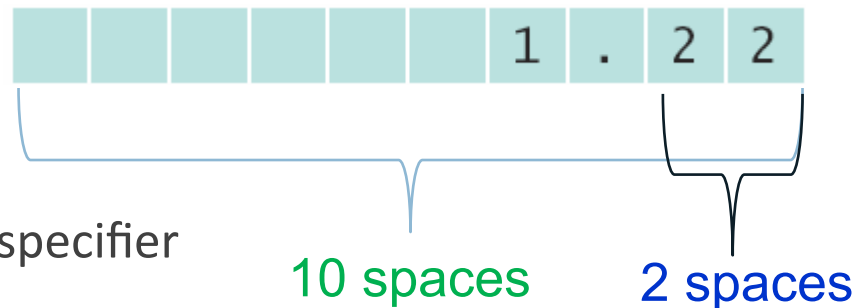
- To control the output appearance of numeric variables, use formatted output tools such as:

```
print("Price per liter %.2f" %(price))
```

Price per liter: 1.22

```
print("Price per liter %10.2f" %(price))
```

Price per liter: 1.22



- The %10.2f is called a format specifier

Syntax: formatting strings

Syntax `formatString % (value1, value2, ..., valuen)`

The format string can contain one or more format specifiers and literal characters.

```
print("Quantity: %d Total: %10.2f" % (quantity, total))
```

It is common to print a formatted string.

Format specifiers

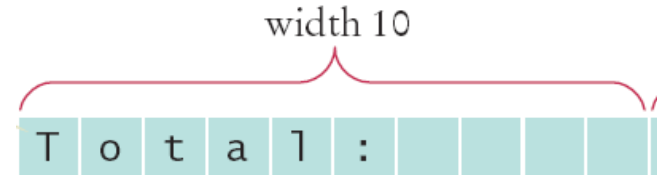
The values to be formatted. Each value replaces one of the format specifiers in the resulting string.

No parentheses are needed to format a single value.

Format flag examples

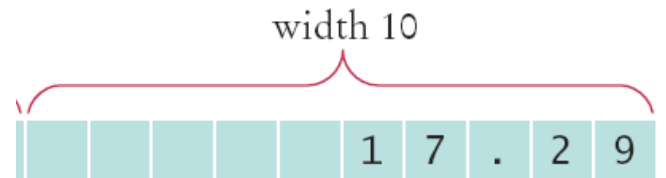
- Left Justify a String:

- `print("%-10s" %("Total:"))`



- Right justify a number with two decimal places

- `print("%10.2f" %(price))`



- And you can print multiple values:

- `print("%-10s%10.2f" %("Total: ", price))`



Volume2.py

ch02/volume2.py

```
1  ##
2  # This program prints the price per ounce for a six-pack of cans.
3  #
4
5  # Define constant for pack size.
6  CANS_PER_PACK = 6
7
8  # Obtain price per pack and can volume.
9  userInput = input("Please enter the price for a six-pack: ")
10 packPrice = float(userInput)
11
12 userInput = input("Please enter the volume for each can (in ounces): ")
13 canVolume = float(userInput)
14
15 # Compute pack volume.
16 packVolume = canVolume * CANS_PER_PACK
17
18 # Compute and print price per ounce.
19 pricePerOunce = packPrice / packVolume
20 print("Price per ounce: %8.2f" % pricePerOunce)
```

Format Specifier Examples

Table 9 Format Specifier Examples		
Format String	Sample Output	Comments
"%d"	2 4	Use d with an integer.
"%5d"	2 4	Spaces are added so that the field width is 5.
"%05d"	0 0 0 2 4	If you add 0 before the field width, zeroes are added instead of spaces.
"Quantity:%5d"	Q u a n t i t y : 2 4	Characters inside a format string but outside a format specifier appear in the output.
"%f"	1 . 2 1 9 9 7	Use f with a floating-point number.
"%.2f"	1 . 2 2	Prints two digits after the decimal point.
"%7.2f"	1 . 2 2	Spaces are added so that the field width is 7.
"%s"	H e l l o	Use s with a string.
"%d %.2f"	2 4 1 . 2 2	You can format multiple values at once.
"%9s"	H e l l o	Strings are right-justified by default.
"%-9s"	H e l l o	Use a negative field width to left-justify.
"%d%"	2 4 %	To add a percent sign to the output, use %.

2.6 Graphics

SIMPLE DRAWINGS

Drawing Simple Graphics

- To help you create simple drawings, we have included a graphics module with the book that is a simplified version of Python's more complex library module.
- The module code and usage instructions are available with the source code for the book on its companion web site.

Using the graphics module (1)

- To create a graphical application using the graphics module, carry out the following at the top of your program:

```
from graphics import GraphicsWindow
```

- Create a graphics window (640 x 480 pixels):

```
win = GraphicsWindow(640, 480)
```

- Access the canvas contained in the graphics window:

```
canvas = win.canvas()
```

Using the graphics module (2)

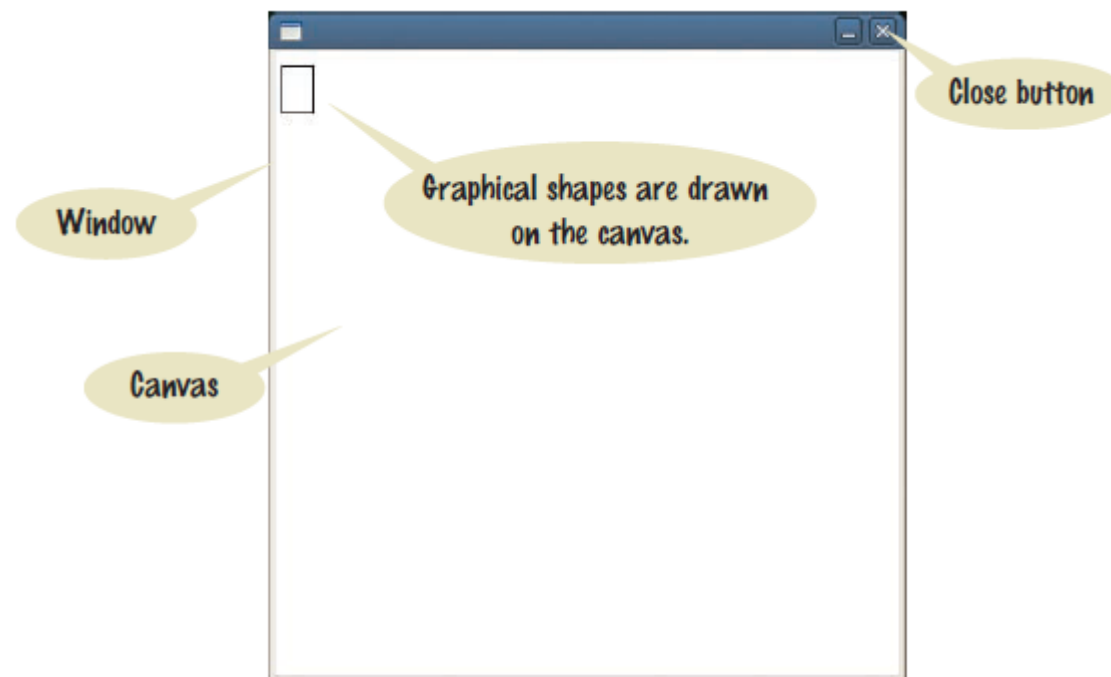
- Create your drawing.

```
canvas.drawRect(15, 10, 20, 30)
```

- Have the program wait for the user to close the window (by clicking the close button).
 - Without this statement, the program would terminate immediately and the graphics window would disappear, leaving no time for you to see your drawing.

```
win.wait()
```

A graphics window



A complete drawing example

ch02/window.py

```
1  ##
2  # This program creates a graphics window with a rectangle. It provides the
3  # template used with all of the graphical programs used in the book.
4  #
5
6  from graphics import GraphicsWindow
7
8  # Create the window and access the canvas.
9  win = GraphicsWindow()
10 canvas = win.canvas()
11
12 # Draw on the canvas.
13 canvas.drawRect(5, 10, 20, 30)
14
15 # Wait for the user to close the window.
16 win.wait()
```

Table 10: GraphicsWindow Methods

Table 10 GraphicsWindow Methods	
Method	Description
$w = \text{GraphicsWindow}()$ $w = \text{GraphicsWindow}(width, height)$	Creates a new graphics window with an empty canvas. The size of the canvas is 400×400 unless another size is specified.
$w.\text{canvas}()$	Returns the object representing the canvas contained in the graphics window.
$w.\text{wait}()$	Keeps the graphics window open and waits for the user to click the “close” button.

Drawing shapes

- Basic shapes have 4 properties: **x coordinate**, **y coordinate**, **width** and **height**.

- Example:

```
canvas.drawRect(15, 10, 20, 30)
```

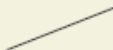



- This statement draws a rectangle with the upper top left corner at point (x = 15, y = 10) in the window with a height of 20 and a width of 30.
- Common shapes that can be drawn include: rectangles, squares, circles and ovals.

Drawing lines

- Lines require 4 slightly different properties than drawing shapes:
 - Point 1(x coordinate, y coordinate)
 - Point 2(x coordinate, y coordinate)

Table 13: Common Shapes, Lines and Text

Table 13 GraphicsCanvas Drawing Methods

Method	Result	Notes
<code>c.drawLine(x₁, y₁, x₂, y₂)</code>		(x_1, y_1) and (x_2, y_2) are the endpoints.
<code>c.drawRect(x, y, width, height)</code>		(x, y) is the top left corner.
<code>c.drawOval(x, y, width, height)</code>		(x, y) is the top-left corner of the box that bounds the ellipse. To draw a circle, use the same value for <i>width</i> and <i>height</i> .
<code>c.drawText(x, y, text)</code>		(x, y) is the anchor point.

The canvas and shapes can be colored

- If you use the default setting (not changing the fill or outline), shapes are outlined in black and there is no fill color.
- The fill color and outline can be set to different colors with the method calls:

```
setFill(<color name>)
```

OR

```
setFill(<red level>, <green level>, <blue level>)
```

```
setOutline(<color name>)
```

OR

```
setOutline(<red level>, <green level>, <blue level>)
```

Example of setting color

- The following statements draw a rectangle that is outlined in black and filled with green.

```
canvas.setOutline("black")
```

```
canvas.setFill(0, 255, 0)
```

```
canvas.drawRect(10, 20, 100, 50)
```



Table 11: Common Color Names

Table 11 Common Color Names			
Color Name	Color Name	Color Name	Color Name
"black"	"magenta"	"maroon"	"pink"
"blue"	"yellow"	"dark blue"	"orange"
"red"	"white"	"dark red"	"sea green"
"green"	"gray"	"dark green"	"light gray"
"cyan"	"gold"	"dark cyan"	"tan"

Table 12: GraphicsCanvas Color Methods

Table 12 GraphicsCanvas Color Methods	
Method	Description
<code>c.setColor(<i>name</i>)</code> <code>c.setColor(<i>red</i>, <i>green</i>, <i>blue</i>)</code>	Sets both the fill and outline color to the same color. Color can be set by the color's <i>name</i> or by values for its <i>red</i> , <i>green</i> , and <i>blue</i> components.
<code>c.setFill()</code> <code>c.setFill(<i>name</i>)</code> <code>c.setFill(<i>red</i>, <i>green</i>, <i>blue</i>)</code>	Sets the color used to fill a geometric shape. If no argument is given, the fill color is cleared.
<code>c.setOutline()</code> <code>c.setOutline(<i>name</i>)</code> <code>c.setOutline(<i>red</i>, <i>green</i>, <i>blue</i>)</code>	Sets the color used to draw lines and text. If no argument is given, the outline color is cleared.

Summary: variables

- A variable is a storage location with a name.
- When defining a variable, you must specify an initial value.
- By convention, variable names should start with a lower case letter.
- An assignment statement stores a new value in a variable, replacing the previously stored value.

Summary: operators

- The assignment operator = does not denote mathematical equality.
- Variables whose initial value should not change are typically capitalized by convention.
- The / operator performs a division yielding a value that may have a fractional value.
- The // operator performs a division, the remainder is discarded.
- The % operator computes the remainder of a floor division.

Summary: python overview

- The Python library declares many mathematical functions, such as `sqrt()` and `abs()`
- You can convert between integers, floats and strings using the respective functions: `int()`, `float()`, `str()`
- Python libraries are grouped into modules. Use the `import` statement to use methods from a module.
- Use the `input()` function to read keyboard input in a console window.

Summary: python overview

- Use the format specifiers to specify how values should be formatted.

Summary: Strings

- Strings are sequences of characters.
- The `len()` function yields the number of characters in a String.
- Use the `+` operator to concatenate Strings; that is, to put them together to yield a longer String.
- In order to perform a concatenation, the `+` operator requires both arguments to be strings. Numbers must be converted to strings using the `str()` function.
- String index numbers are counted starting with 0.

Summary: Strings

- Use the [] operator to extract the elements of a String.

Summary: graphics

- Graphical shapes (such as squares, rectangles, circles, ovals), or lines and text can be drawn using the graphics module.
- The color of graphical objects can be set with the `setOutline()` and `setFill()` methods.

Chapter Goals

- To declare and initialize variables and constants
- To understand the properties and limitations of integers and floating-point numbers
- To appreciate the importance of comments and good code layout
- To write arithmetic expressions and assignment statements
- To create programs that read, and process inputs, and display the results
- To learn how to use Python strings
- To create simple graphics programs using basic shapes and text