Exposure CS 2021 for CS1

Chapter 6 Section 9-10 Slides

More Python Libraries: Displaying Graphics Text & Subroutine Review

PowerPoint Presentation created by:
Mr. John L. M. Schram and Mr. Leon Schram
Authors of Exposure Computer Science

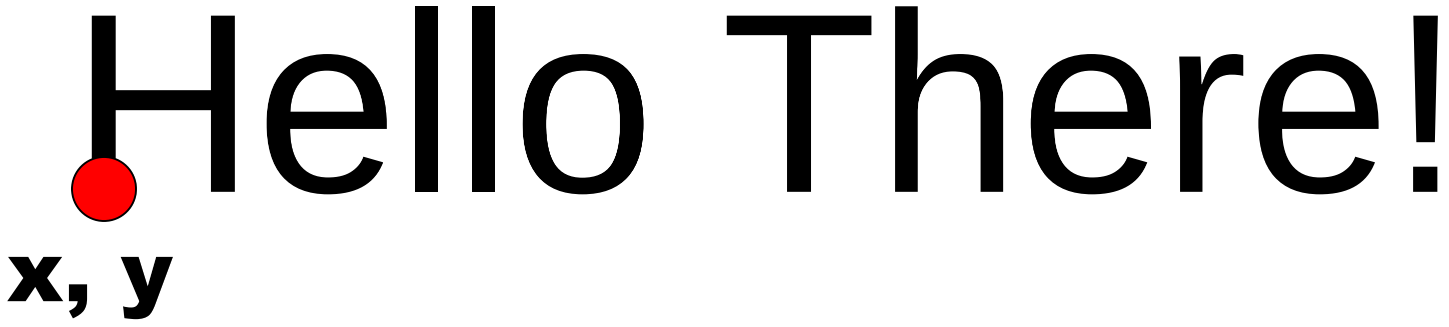
Section 6.9

Displaying Graphics Text

Procedure drawString

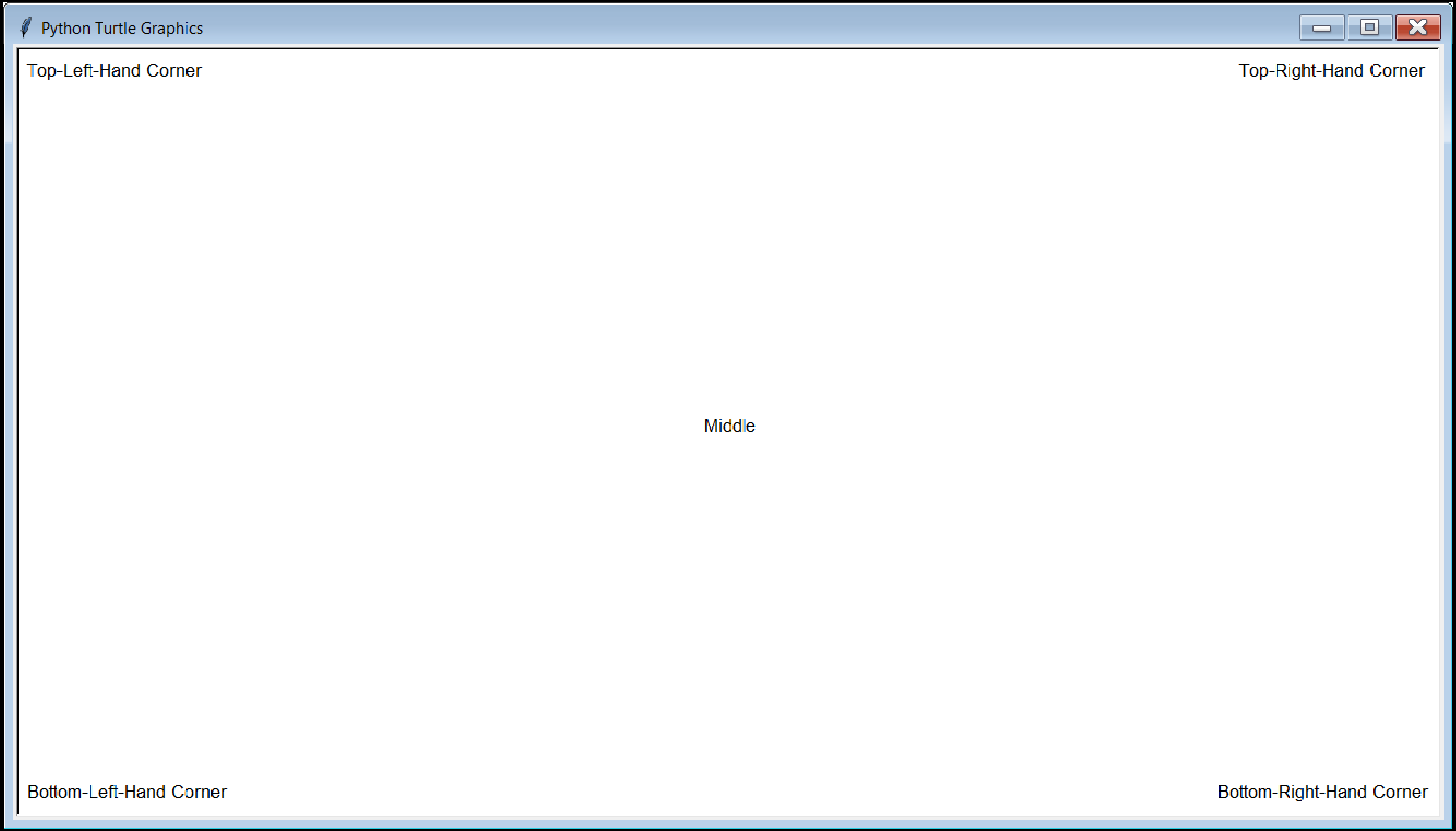
```
drawString("Hello There!", x, y)
```

Draws any string starting at coordinate (x,y).



```python
# GraphicsLibrary25.py
# This program demonstrates the <drawString>
# procedure of the <Graphics> library.  With
# <drawString("Hello World",x,y)>, the string
# "Hello World" will be displayed starting at
# coordinate (x,y).


from Graphics import *

beginGrfx(1300,700)

drawString("Top-Left-Hand Corner",10,30)
drawString("Top-Right-Hand Corner",1120,30)
drawString("Bottom-Left-Hand Corner",10,690)
drawString("Bottom-Right-Hand Corner",1100,690)
drawString("Middle",630,355)

endGrfx()
```

GraphicsLibrary25.py Output

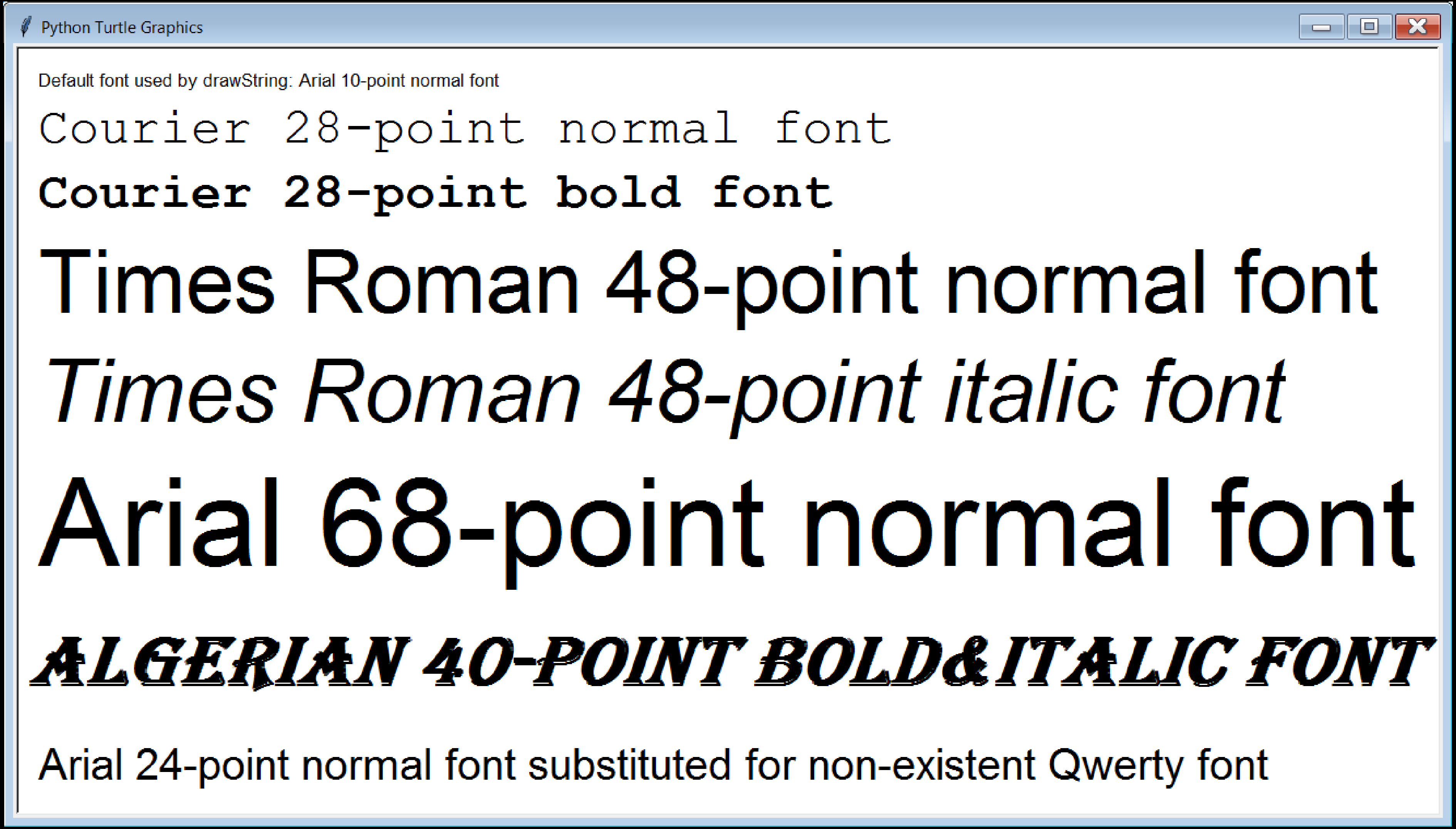


Python Turtle Graphics

```
Top-Left-Hand Corner                                          Top-Right-Hand Corner

                                 Middle

Bottom-Left-Hand Corner                                    Bottom-Right-Hand Corner
```

```python
# GraphicsLibrary26.py
# This program demonstrates that the <drawString>
# procedure can have up to a total of 6 arguments.
# The optional 4th, 5th and 6th arguments specify
# the "Font Face" (name of the font), the "Font Size"
# and the "Font Style" which can be <"normal">,
# <"bold">, <"italic"> or <"bold&italic">.


from Graphics import *

beginGrfx(1300,700)

drawString("Default font used by drawString: Arial 10-point normal font",20,40)
drawString("Courier 28-point normal font",20,100, "Courier",28,"normal")
drawString("Courier 28-point bold font",20,160, "Courier",28,"bold")
drawString("Times  Roman  48-point  normal  font",20,  260,"TimesRoman",48)  # "normal" is default
drawString("Times  Roman  48-point  italic  font",20,  360,"TimesRoman",48,"italic")
drawString("Arial 68-point normal font",20,500, "Arial",68)
drawString("Algerian 40-point bold&italic font ",10, 600,"Algerian",40,"bold&italic")
drawString("Arial 24-point normal font substituted for non-existent Qwerty font",20,680,
"Qwerty",24)

endGrfx()
```

GraphicsLibrary26.py Output



Default font used by drawString: Arial 10-point normal font

Courier 28-point normal font

Courier 28-point bold font

Times Roman 48-point normal font

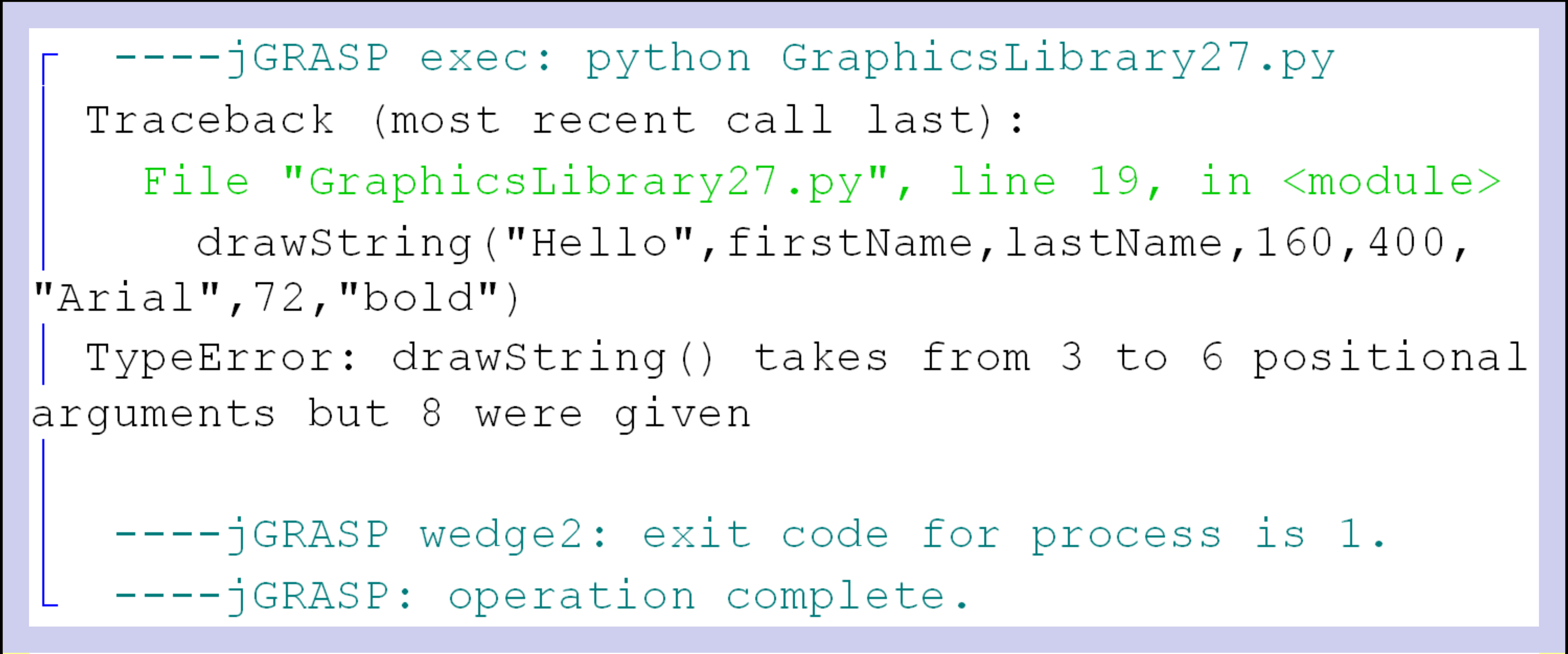
Times Roman 48-point italic font

Arial 68-point normal font

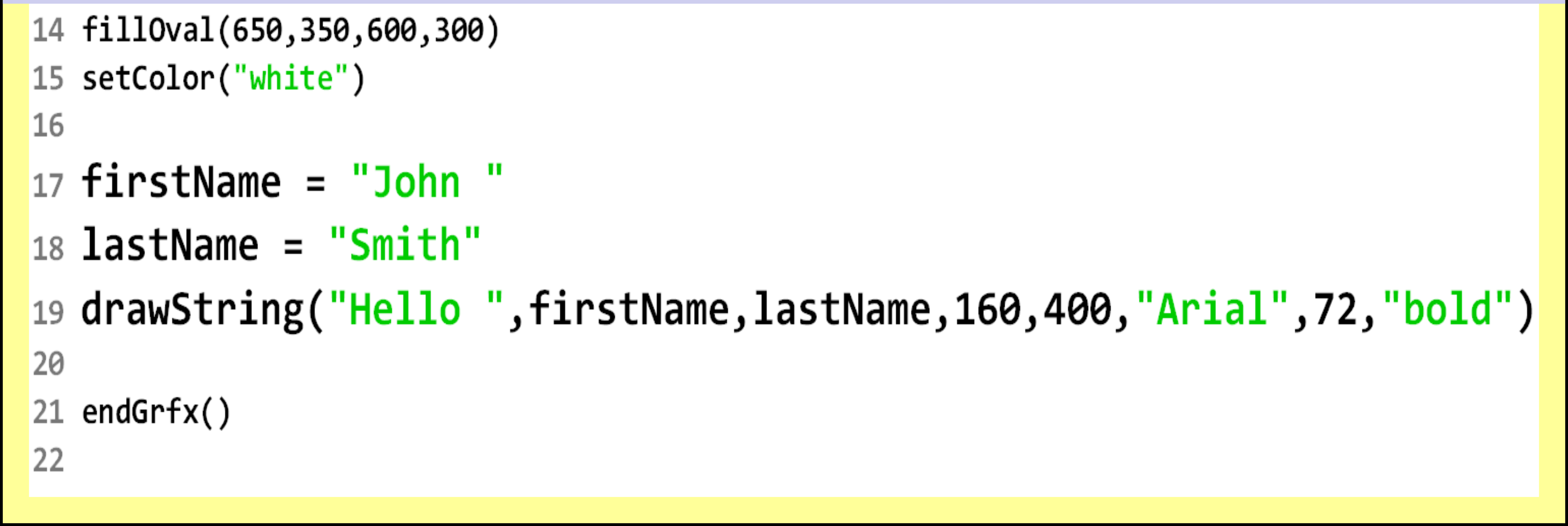
ALGERIAN 40-POINT BOLD&ITALIC FONT

Arial 24-point normal font substituted for non-existent Qwerty font

```python
# GraphicsLibrary27.py
# This program demonstrates what happens when you try to
# display multiple pieces of information with <drawString>
# separated with commas like you do with <print>.
# This does not work because the each separate piece of
# information is its own argument.


from Graphics import *

beginGrfx(1300,700)

setColor("red")
fillOval(650,350,600,300)
setColor("white")

firstName = "John "
lastName = "Smith"
drawString("Hello ",firstName,lastName,160,400,"Arial",72,"bold")

endGrfx()

```



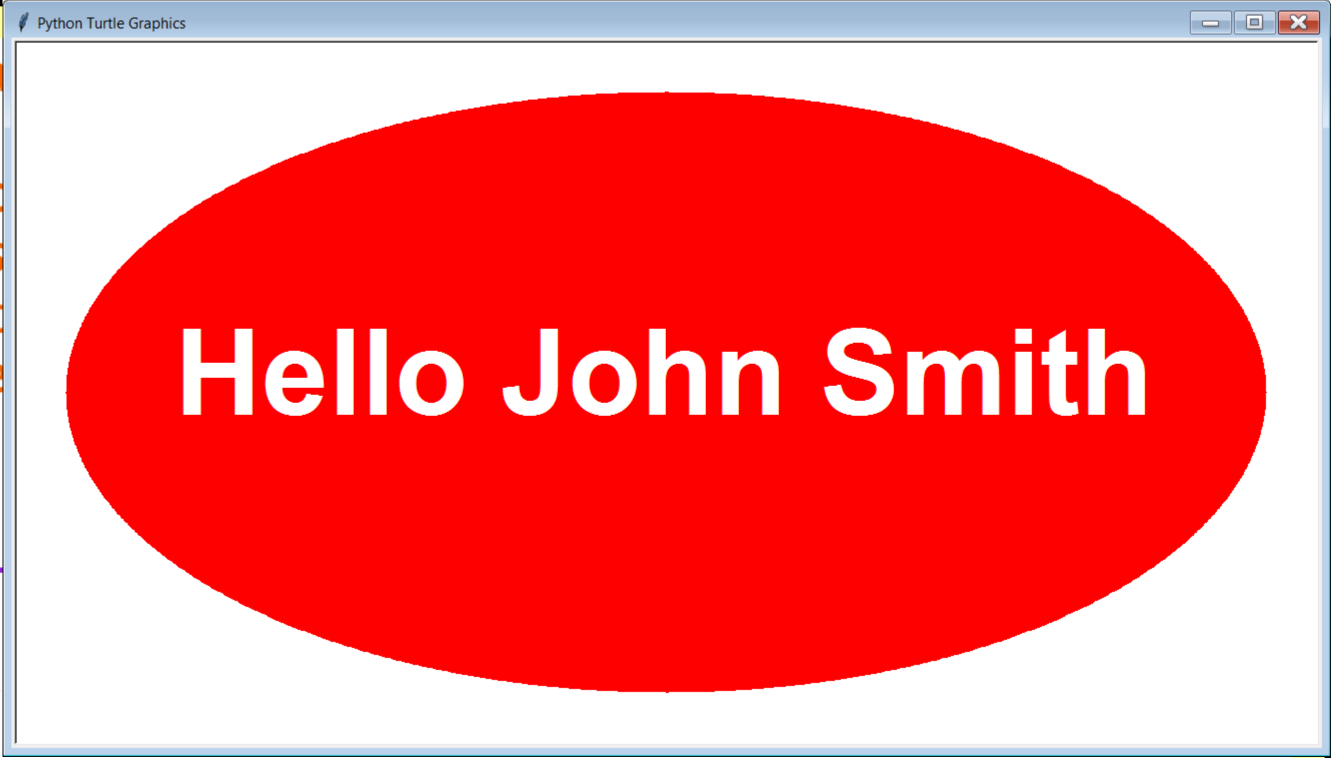
```
  ----jGRASP exec: python GraphicsLibrary27.py
 Traceback (most recent call last):
   File "GraphicsLibrary27.py", line 19, in <module>
     drawString("Hello",firstName,lastName,160,400,
"Arial",72,"bold")
 TypeError: drawString() takes from 3 to 6 positional
arguments but 8 were given


  ----jGRASP wedge2: exit code for process is 1.
  ----jGRASP: operation complete.
```



```
14 fillOval(650,350,600,300)
15 setColor("white")
16
17 firstName = "John "
18 lastName = "Smith"
19 drawString("Hello ",firstName,lastName,160,400,"Arial",72,"bold")
20
21 endGrfx()
22
```

```python
# GraphicsLibrary28.py
# This program demonstrates the proper way to display
# multiple pieces of information with <drawString>.
# The secret is to use String Concatenation to combine
# the different pieces of information into a single
# string argument.


from Graphics import *

beginGrfx(1300,700)

setColor("red")
fillOval(650,350,600,300)
setColor("white")

firstName = "John "
lastName = "Smith"
drawString("Hello "+firstName+lastName,160,400,
"Arial",72,"bold")

endGrfx()
```

```
# GraphicsLibra
# This program
# multiple piec
# The secret is
# the different
# string argume


from Graphics i

beginGrfx(1300,

setColor("red")
fillOval(650,350,600,300)
setColor("white")

firstName = "John "
lastName = "Smith"
drawString("Hello "+firstName+lastName,160,400,
"Arial",72,"bold")

endGrfx()
```



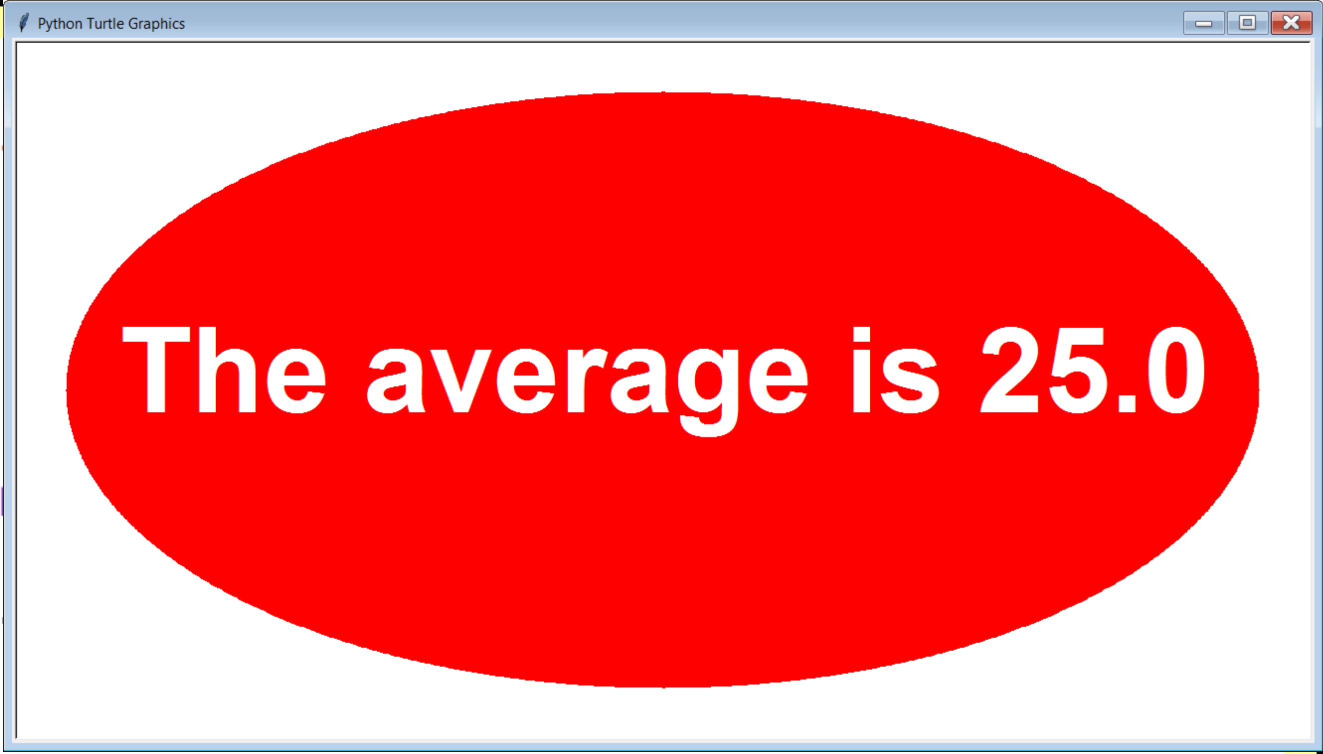
```python
# GraphicsLibrary29.py
# This program demonstrates that the concatenation
# trick does not work if one of the pieces of
# information is a number.


from Graphics import *

beginGrfx(1300,700)

setColor("red")
fillOval(650,350,600,300)
setColor("white")

average = (10 + 20 + 30 + 40) / 4
drawString("The average is "+average,105,400,"Arial",72,"bold")

endGrfx()
```

```
 ----jGRASP exec: python GraphicsLibrary29.py
Traceback (most recent call last):
  File "GraphicsLibrary29.py", line 16, in <module>
    drawString("The average is "+average,160,400,
"Arial",72,"bold")
TypeError: must be str, not float

 ----jGRASP wedge2: exit code for process is 1.
 ----jGRASP: operation complete.
```

```
11 setColor("red")
12 fillOval(650,350,600,300)
13 setColor("white")
14
15 average = (10 + 20 + 30 + 40) / 4
16 drawString("The average is "+average,105,400,
"Arial",72,"bold")
17
18 endGrfx()
```

```python
# GraphicsLibrary30.py
# This program fixes the problem of the previous program
# by using <str> to convert the number to a string.
# Now it can be concatenated with other strings.


from Graphics import *

beginGrfx(1300,700)

setColor("red")
fillOval(650,350,600,300)
setColor("white")

average = (10 + 20 + 30 + 40) / 4
drawString("The average is "+str(average),
105,400,"Arial",72,"bold")

endGrfx()
```

```
# GraphicsLibra
# This program
# by using <str
# Now it can be


from Graphics i

beginGrfx(1300,

setColor("red")
fillOval(650,350,600,300)
setColor("white")

average = (10 + 20 + 30 + 40) / 4
drawString("The average is "+str(average),
105,400,"Arial",72,"bold")

endGrfx()
```



```
The average is 25.0
```

Section 6.10

Review: Functions vs. Procedures

Subroutines, Functions and Procedures Review

A *subroutine* is a series of programming commands that performs a specific task.

A *function* is a subroutine that returns a value.

The subroutines in the **math** library are *functions*.

A *procedure* is a subroutine that does not return a value.

The subroutines in the **Graphics** library are mostly *procedures*.