Parameters

def \texttt{name}(\texttt{parameter, parameter, ..., parameter}): statements

- Parameters are declared by writing their names (no types)

\begin{verbatim}
print_many.py
def print_many(\texttt{word, n}):
    for i in range(\texttt{n}):
        print \texttt{word}
print_many(\texttt{"hello",4})

>>> hello
hello
hello
hello
\end{verbatim}
Exercise

- Write a function `box(width, height)` and print the following patterns by calling the function, for example,

```python
box(10,3)
box(5,4)
```

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

```python

```
# Draws a box of stars with the given width and height.

```python
def box(width, height):
    print (width * "*"")
    for i in range(height - 2):
        print ("*" + (width - 2) * " " + "*")
    print (width * "*"")

# main
box(10, 3)
box(5, 4)
```
Default Parameter Values

def name(parameter=value, ..., parameter=value):
    statements

- Can make parameter(s) optional by specifying a default value

>>> def print_many(word, n=1):
...     for i in range(n):
...         print word

>>> print_many("shrubbery")
shrubbery
>>> print_many("shrubbery", 4)
shrubbery
shrubbery
shrubbery
shrubbery

- Exercise: Modify stars.py to add an optional parameter for the character to use for the outline of the box (default "*").
Parameter Keywords

name (parameter=value, ..., parameter=value)

- Can specify the names of parameters as you call a function
- This allows you to pass the parameters in any order

```python
>>> def print_many(word, n):
...     for i in range(n):
...         print word
...     print word

>>> print_many(word="shrubbery", n=4)
shrubbery
shrubbery
shrubbery
shrubbery
>>> print_many(n=3, word="Ni!")
Ni!
Ni!
Ni!
```
from math import *

<table>
<thead>
<tr>
<th>Function name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceil(value)</td>
<td>rounds up</td>
</tr>
<tr>
<td>cos(value)</td>
<td>cosine, in radians</td>
</tr>
<tr>
<td>degrees(value)</td>
<td>convert radians to degrees</td>
</tr>
<tr>
<td>floor(value)</td>
<td>rounds down</td>
</tr>
<tr>
<td>log(value, base)</td>
<td>logarithm in any base</td>
</tr>
<tr>
<td>log10(value)</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>max(value1, value2, ...)</td>
<td>largest of two (or more) values</td>
</tr>
<tr>
<td>min(value1, value2, ...)</td>
<td>smallest of two (or more) values</td>
</tr>
<tr>
<td>radians(value)</td>
<td>convert degrees to radians</td>
</tr>
<tr>
<td>round(value)</td>
<td>nearest whole number</td>
</tr>
<tr>
<td>sin(value)</td>
<td>sine, in radians</td>
</tr>
<tr>
<td>sqrt(value)</td>
<td>square root</td>
</tr>
<tr>
<td>tan(value)</td>
<td>tangent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>pi</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>
Exercise

Write a simple program to find the hypotenuse \( c \) of a right triangle given sides \( a \) and \( b \).
from math import *

a=3
b=4
c=sqrt(3**2 + 4**2)

print('c=',c)

# let's change a=-3,
a=-3
c=sqrt(3**2 + 4**2)

print('c=',c)

# how to fix it? (how to check a, b to make sure >0)
def name(parameters):
    statements

    return value

def ftoc(temp):
    tempc = 5.0 / 9.0 * (temp - 32)
    return tempc

print(ftoc(98.6))

Write a Python code to convert the Celsius to Fahrenheit. Define a function and Call this function to convert 24, 30 Celsius to Fahrenheit.
**DrawingPanel**

- Instructor-provided `drawingpanel.py` file must be in the same folder as your Python program

- At the top of your program, write:
  
  ```python
  from drawingpanel import *
  ```

- Panel's `canvas` field behaves like `Graphics`
```python
draw1.py

1 from drawingpanel import *
2 3 panel = DrawingPanel(400, 300)
4 panel.set_background("yellow")
5 panel.canvas.create_rectangle(100, 50, 200, 300)
```

Left upper corner (0,0)
Right lower corner(399,299)
## Drawing Methods

<table>
<thead>
<tr>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>panel.canvas.create_line(x1, y1, x2, y2)</code></td>
</tr>
<tr>
<td><code>panel.canvas.create_rectangle(x1, y1, x2, y2)</code></td>
</tr>
<tr>
<td><code>panel.canvas.create_oval(x1, y1, x2, y2)</code></td>
</tr>
<tr>
<td><code>panel.canvas.create_text(x, y, text=&quot;text&quot;)</code></td>
</tr>
<tr>
<td><em>(see next slide)</em></td>
</tr>
<tr>
<td><code>panel.set_background(color)</code></td>
</tr>
</tbody>
</table>

- Notice, methods take x2,y2 parameters, not width/height
Colors and Fill

- Python doesn't have `fillRect`, `fillOval`, or `setColor`. Instead, pass outline and fill colors when drawing a shape.
- List of all color names: [http://wiki.tcl.tk/16166](http://wiki.tcl.tk/16166)
- Visual display of all colors

```python
from drawingpanel import *

panel = DrawingPanel(400, 300)
panel.canvas.create_rectangle(100, 50, 200, 200, outline="red", fill="yellow")
panel.canvas.create_oval(20, 10, 180, 70, fill="blue")
```
• Draw arbitrary polygons with `create_polygon`
• Draw line groups by passing more params to `create_line`

drawpoly.py

```
from drawingpanel import *
panel = DrawingPanel(200, 200)
panel.canvas.create_polygon(100, 50, 150, 0, 150, 100, fill="green")
panel.canvas.create_line(10, 120, 20, 160, 30, 120, 40, 175)
```
Exercise

Let’s create a car in Python:

Car: width = 100 pix, height = 50 pix
Exercise

Draw a car in Python
Exercise

Now, let’s use parameters so that we can place the cars all over the DrawingPanel.

i.e., define a function to draw a car and then call this function to draw the function in different places.
Exercise

Animate it using `panel.sleep()`

```
panel.sleep(200),
```

→ sleep 200 mini seconds