CSCI 141

Scott Wehrwein

(Tuples,)
Goals

• Understand the basic usage of **tuples**:
  
  • using tuples to return multiple values from a function
  
  • **packing** and **unpacking** via the assignment operator
Your task: Draw this.
# Let p be a random point in the window
# loop 10000 times:
#     c = a random corner of the triangle
#     m = the midpoint between p and c
#     choose a color for m
#     color the pixel at m
#     p=m

Strategy: break this down into manageable pieces by inventing functions that solve pieces of the problem!
# Let p be a random point in the window
# loop 10000 times:
#   c = a random corner of the triangle
#   m = the midpoint between p and c
#   choose a color for m
#   color the pixel at m
#   p=m
Midpoint Function

```python
def midpoint(p1x, p1y, p2x, p2y):
    """Return the midpoint between (p1x, p1y) and (p2x, p2y)"
    """
    # code here
```
Midpoint Function

def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here
def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here

(mid_x, mid_y)
def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here

(mid_x, mid_y)

This is two things!? Can we return two things?
def midpoint(p1x, p1y, p2x, p2y):
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) ""
    # code here
    # mid_x = . . .
    # mid_y = . . .
    return mid_x, mid_y
Returning Multiple Values

- You can return multiple values from a function by grouping them into a comma-separated sequence:
  
  ```python
  return mid_x, mid_y
  ```

- You can assign each to a variable when calling the function like this:
  
  ```python
  mx, my = midpoint(plx, ply, p2x, p2y)
  ```
These are actually **tuples**

- A tuple is a sequence of values, optionally enclosed in parens.
  
  \[(1, 4, "Mufasa")\]

- You can “pack” and “unpack” them using assignment statements:
  
  ```
  v = (1, 4, "Mufasa") # packing
  (a, b, c) = v # "unpacking"
  ```
These are actually **tuples**

- Tuples can also be passed *into* functions as arguments:

```python
def midpoint(p1, p2):
    """Compute the midpoint between p1 and p2"""
    p1x, p1y = p1
    p2x, p2y = p2

    # . . .
    # return mx, my
```
Tuples: Demo
Tuples: Demo

• assignment, packing, unpacking

• with and without parens (printing)

• swapping

• equality

• mismatched # values to unpack
Tuples - 1

\[
\begin{align*}
a &= 1 \\
b &= 2 \\
c &= 3 \\
v &= (a, a, c) \\
\text{print}(v, \text{sep}="\ ")
\end{align*}
\]

# What does this print?
# A: 1 2 3
# B: 1 1 3
# C: (1, 2, 3)
# D: (1, 1, 3)
Tuples - 2

a = 1
b = 2
c = 3

a, b, c = (a, a, c)

print(a, b, c, sep=" ")

# What does this print?
# A: 1 2 3
# B: 1 1 3
# C: (1, 2, 3)
# D: (1, 1, 3)
def midpoint(p1x, p1y, p2x, p2y):
    
    """ Return the midpoint between (p1x, p1y) and (p2x, p2y) """

    # code here
    # mid_x = . . .
    # mid_y = . . .

    return mid_x, mid_y
Midpoint Function

Okay, but how do you actually calculate this?

\[
\begin{align*}
\text{mid}_x &= \frac{p1_x + p2_x}{2} \\
\text{mid}_y &= \frac{p1_y + p2_y}{2}
\end{align*}
\]
Demo: writing the midpoint function

• With tuple as return value

• Switch to tuples as parameters for points