Code Library · Stage 9

Six tiny programs about HELPER FUNCTIONS - reusable named blocks of code. Copy, paste, tweak.

First, a tiny bit of background

A **function** is a recipe with a name. You write the steps once with def, then call the name whenever you want those steps to run. Functions can take **parameters** (ingredients you pass in) and have **defaults** for when you don't.

Inside a function, if you want to change a variable that lives OUTSIDE the function, write global variable_name at the top. Without that, Python thinks you're making a new local variable trapped in the function.

```
# the heart of Stage 9
def draw_text(text, x, y, color=WHITE, big=False):
    f = big_font if big else font
    screen.blit(f.render(text, True, color), (x, y))

def reset_game():
    global score, lives, bullets
    score, lives, bullets = 0, 3, []

# call them anywhere - one line each
draw_text("Score: " + str(score), 10, 10)
reset_game()
```

How to use this handout

Each game has three parts: **what it does**, **the code**, and **challenges**. Copy any code block into the Trinket editor on the lesson page, press the green ■ **Run** button, then add a parameter, change a default, or call your function from a new place - see what happens.

Game 1 · Hello Function

What it does

The simplest function in Python. We define greet(), then call it three times. Three lines of output from a one-line recipe. (No pygame - just a console program.)

```
# the simplest function in Python

def greet():
    print("Hello, pilot!")

greet()
greet()
greet()
print("---")
print("Three hellos with one recipe.")
```

Try these one by one

Hello 1 Add a fourth call to greet() and a print() after it.

```
greet()
print("That makes four!")
```

Hello 2 Change the message to your name.

```
def greet():
    print("Hi, this is Goodness!")
```

Hello 3 Make greet() print two lines instead of one.

```
def greet():
    print("Hello, pilot!")
    print("Welcome aboard!")
```

Game 2 · Greet with a Name (parameters)

What it does

Adds an ingredient (parameter) to the function so it can greet anyone. Then a second function shows what a DEFAULT parameter looks like - if you don't pass a name, it uses "pilot" automatically.

```
# functions get more useful when you can pass them ingredients (parameters)

def greet(name):
    print("Hello, " + name + "!")

greet("Aisha")
greet("Tomi")
greet("Lakshmi")

# you can also give parameters a DEFAULT, used when caller skips it
def cheer(name="pilot"):
    print("Go, " + name + ", go!")

cheer()                 # uses default -> Go, pilot, go!
cheer("Captain Nova")
```

Try these one by one

Param 1 Add greet("Mum") and greet("Dad") to the list.

Param 2 Change cheer to default to "team".

```
def cheer(name="team"):
```

Param 3 Add a SECOND parameter: def greet(name, age): then print both.

```
print("Hello, " + name + "! Age " + str(age))
```

Game 3 · draw_text Helper (the real Galaxy Defender helper)

What it does

This is THE helper from Galaxy Defender. Three lines (pick font, render image, blit it) wrapped into one tidy draw_text(text, x, y, color, big). Look at how clean the inner loop becomes - four pieces of text in four readable lines.

```
import pygame

pygame.init()
screen = pygame.display.set_mode((600, 400))
pygame.display.set_caption("draw_text Helper")

WHITE  = (255, 255, 255)
YELLOW = (255, 215, 0)
RED    = (255, 0, 0)
SPACE  = (0, 0, 50)

font     = pygame.font.SysFont("Arial", 32)
big_font = pygame.font.SysFont("Arial", 64, bold=True)

# THE HELPER - three lines wrapped into one neat function
def draw_text(text, x, y, color=WHITE, big=False):
    f = big_font if big else font
    img = f.render(text, True, color)
    screen.blit(img, (x, y))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: pygame.quit(); exit()

    screen.fill(SPACE)

    # look how short this is! one line per piece of text
    draw_text("Score: 42", 20, 20)
    draw_text("Lives: 3",  20, 60, color=YELLOW)
    draw_text("WARNING!",  20, 110, color=RED)
    draw_text("BIG NEWS",  20, 170, color=YELLOW, big=True)

    pygame.display.flip()
```

Try these one by one

Draw 1 Add a fifth piece of text in green at the bottom of the screen.

```
GREEN = (0, 220, 100)
draw_text("OK!", 20, 260, color=GREEN)
```

Draw 2 Default the color to YELLOW instead of WHITE.

```
def draw_text(text, x, y, color=YELLOW, big=False):
```

Draw 3 Add a parameter italic=False. If True, use a different font that's italic.

Game 4 · Reset Helper (global keyword)

What it does

Press SPACE to score. Press R to reset - the score returns to zero, BUT the high score is remembered. reset_game() is one named action that bundles the entire reset together. The magic word global tells Python which outside variables you mean.

```
import pygame

pygame.init()
screen = pygame.display.set_mode((600, 400))
pygame.display.set_caption("Reset Helper - press R to reset, SPACE to score")

WHITE = (255, 255, 255)
SPACE = (0, 0, 50)
font  = pygame.font.SysFont("Arial", 48, bold=True)

score = 0
high  = 0

# THE HELPER - one named action: "reset"
def reset_game():
    global score        # we want to change the OUTER score, not make a new one
    if score > globals().get("high", 0):
        globals()["high"] = score
    score = 0

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: pygame.quit(); exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE: score += 1
            if event.key == pygame.K_r:     reset_game()

    screen.fill(SPACE)
    img1 = font.render("Score: " + str(score), True, WHITE)
    img2 = font.render("HIGH:  " + str(high),  True, (255, 215, 0))
    screen.blit(img1, (60, 100))
    screen.blit(img2, (60, 180))
    pygame.display.flip()
```

Try these one by one

Reset 1 Make R also reset the high score (remove the protect-the-high check).

```
def reset_game():
    global score, high
    score = 0
    high = 0
```

Reset 2 Add print("Game reset!") inside the function.

Reset 3 Make SPACE earn 5 points each press.

```
if event.key == pygame.K_SPACE: score += 5
```

Game 5 · Spawn Enemy Helper (return values)

What it does

Press SPACE and a brand-new enemy appears at a random spot above the screen. make_enemy() doesn't print anything - it RETURNS a value (a Rect) using the return keyword. We catch the returned value with enemies.append(make_enemy()).

```
import pygame
import random

pygame.init()
WIDTH, HEIGHT = 600, 400
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Spawn Enemy Helper - press SPACE to add one")
clock = pygame.time.Clock()

RED   = (240, 60, 60)
SPACE = (0, 0, 50)

enemies = []

# THE HELPER - returns a brand-new Rect at a random spot
def make_enemy():
    return pygame.Rect(random.randint(0, WIDTH - 40),
                       random.randint(-200, 0),
                       40, 40)

# start with three enemies
for _ in range(3):
    enemies.append(make_enemy())

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT: pygame.quit(); exit()
        if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
            enemies.append(make_enemy())     # one tidy line!

    for e in enemies:
        e.y += 3
        if e.y > HEIGHT:
            e.y = random.randint(-200, -50)
            e.x = random.randint(0, WIDTH - 40)

    screen.fill(SPACE)
    for e in enemies: pygame.draw.rect(screen, RED, e)
    pygame.display.flip()
    clock.tick(60)
```

Try these one by one

Spawn 1 Start with 10 enemies instead of 3.

```
for _ in range(10):
```

Spawn 2 Make make_enemy() take a width parameter: def make_enemy(width=40):

```
def make_enemy(width=40):
    return pygame.Rect(..., ..., width, 40)
```

Spawn 3 Spawn an enemy every time the MOUSE is clicked instead of SPACE.

```
if event.type == pygame.MOUSEBUTTONDOWN:
```

Game 6 · Mini Defender with Helpers

The complete Galaxy Defender REFACTORED with three helpers: **draw_text()**, **make_enemy()**, **reset_game()**. Same game as before, but the code reads like a recipe instead of a wall of spaghetti.

Part 1 of 2 - setup & helpers:

```
import pygame
import random

pygame.init()
WIDTH, HEIGHT = 800, 500
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Mini Defender with Helpers")
clock = pygame.time.Clock()

GREEN, YELLOW, RED, WHITE, SPACE = (0,220,100),(255,215,0),(255,0,0),(255,255,255),(0,0,50)
font = pygame.font.SysFont("Arial", 28)
big  = pygame.font.SysFont("Arial", 60, bold=True)

player, bullets, enemies, score, state = pygame.Rect(370,420,60,40), [], [], 0, "PLAYING"

# ■■ 3 HELPER FUNCTIONS ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
def draw_text(text, x, y, color=WHITE, big_=False):
    f = big if big_ else font
    screen.blit(f.render(text, True, color), (x, y))

def make_enemy():
    return pygame.Rect(random.randint(0, WIDTH-50),
                       random.randint(-300, 0), 50, 50)

def reset_game():
    global score, bullets, state
    score, bullets, state = 0, [], "PLAYING"
    player.x = 370
    for i in range(len(enemies)):
        enemies[i] = make_enemy()

for _ in range(5):
    enemies.append(make_enemy())

# ■■ game loop continues on the next page... ■■
```

Game 6 · Mini Defender with Helpers (continued)

Part 2 of 2 - the game loop:

```
# ■■ ...continued from the previous page ■■

while True:
    for ev in pygame.event.get():
        if ev.type == pygame.QUIT: pygame.quit(); exit()
        if ev.type == pygame.KEYDOWN:
            if ev.key == pygame.K_SPACE and state == "PLAYING":
                bullets.append(pygame.Rect(player.x+27, player.y, 6, 15))
            if ev.key == pygame.K_r and state == "GAME_OVER":
                reset_game()

    keys = pygame.key.get_pressed()
    screen.fill(SPACE)

    if state == "PLAYING":
        if keys[pygame.K_LEFT]  and player.x > 0:          player.x -= 5
        if keys[pygame.K_RIGHT] and player.x < WIDTH - 60: player.x += 5

        for b in bullets[:]:
            b.y -= 10
            if b.y < 0: bullets.remove(b)

        for e in enemies:
            e.y += 2
            if e.y > HEIGHT:
                e.y, e.x = random.randint(-200,-50), random.randint(0, WIDTH-50)
            if player.colliderect(e): state = "GAME_OVER"

        for b in bullets[:]:
            for e in enemies:
                if b.colliderect(e):
                    if b in bullets: bullets.remove(b)
                    e.y, e.x = random.randint(-200,-50), random.randint(0, WIDTH-50)
                    score += 1

        pygame.draw.rect(screen, GREEN, player)
        for b in bullets: pygame.draw.rect(screen, YELLOW, b)
        for e in enemies: pygame.draw.rect(screen, RED, e)
        draw_text("Score: " + str(score), 10, 10)

    else:  # GAME_OVER
        draw_text("GAME OVER", WIDTH//2 - 170, 180, color=RED, big_=True)
        draw_text("Press R to restart", WIDTH//2 - 105, 280, color=YELLOW)

    pygame.display.flip()
    clock.tick(60)
```

Try these one by one

Mini 1 Add a fourth helper: def check_hits() that contains the bullet-vs-enemy loop. Use global score inside.

Mini 2 Add a MENU state (from Lesson 8) and a draw_menu() helper.

Mini 3 Print a list of all your helpers at the top of the file as comments. (Helps anyone reading your code instantly understand the structure.)

```
# HELPERS: draw_text, make_enemy, reset_game
```

Where this takes us next

You've now reached the secret of professional code: **don't repeat yourself**. Pros call this DRY. Your future self will thank you when you can read a function name and instantly know what it does. Lesson 10 - the LAST stage - zooms back out to the GAME LOOP itself: how the 60-frames-per-second heartbeat ties everything together.

Challenge: take any of your earlier games and rewrite ONE repeated chunk as a helper function. You'll feel the difference immediately.