Functions

Writing Reusable Code

What Is a Function?

A function is a way to **group code** that does something specific, so you can use it multiple times without rewriting it.

You've already been using functions!

- `print('Hello')` - `print` is a function that displays text
- `input('Enter name: ')` - `input` is a function that gets user input
- `int('42')` - `int` is a function that converts its input to an integer
- `len('Python')` - `len` is a function that gets the length of a string
- `range(5)` - `range` is a function that generates a sequence of numbers
- `randint(1, 10)` - `randint` is a function that generates a random integer

A function is a **named block of code** that does a specific task.

Why Use Functions?

Instead of copying code:

```
# Calculate tip for lunch
lunch_bill = 25.00
lunch_tip = lunch_bill * 0.20
print(f"Lunch tip: ${lunch_tip}")

# Calculate tip for dinner (same code!)
dinner_bill = 45.00
dinner_tip = dinner_bill * 0.20
print(f"Dinner tip: ${dinner_tip}")
```

```
Lunch tip: $5.0
Dinner tip: $9.0
```

We can write it once and reuse it!

```
def calculate_tip(bill, tip_percent=0.20):
    tip = bill * tip_percent
    return tip

# Calculate tip for lunch
lunch_bill = 25.00
lunch_tip = calculate_tip(lunch_bill)
print(f"Lunch tip: ${lunch_tip}")

# Calculate tip for dinner
dinner_bill = 45.00
dinner_tip = calculate_tip(dinner_bill)
print(f"Dinner tip: ${dinner_tip}")
```

```
Lunch tip: $5.0
Dinner tip: $9.0
```

Your First Function

```python
def say_hello():
    print("Hello!")
    print("Welcome to Python!")

# Call the function
say_hello()
```

```
Hello!
Welcome to Python!
```

Key parts:

- `def` keyword starts a function
- Function name followed by `()`
- Colon `:`
- Indented code is the function body

Calling Functions

Writing a function doesn't run it. You must **call** it:

```
def greet():
    print("Good morning!")

# Function exists but hasn't run yet

greet()  # NOW it runs!
greet()  # We can call it multiple times
greet()  # Each call runs the code inside
```

```
Good morning!
Good morning!
Good morning!
```

Functions Can Take Input

Functions can accept **arguments** (input values):

```
def greet_person(name):
    print(f"Hello, {name}!")
    print(f"Nice to meet you, {name}")

# Call with different names
greet_person("Alice")
greet_person("Bob")
```

```
Hello, Alice!
Nice to meet you, Alice
Hello, Bob!
Nice to meet you, Bob
```

The `name` is a **parameter** - it holds whatever value we pass in

Multiple Parameters

Functions can take multiple inputs:

```
def calculate_tip(bill, tip_percent):
    tip = bill * (tip_percent / 100)
    total = bill + tip
    print(f"Bill: ${bill}")
    print(f"Tip: ${tip}")
    print(f"Total: ${total}")

# Use it for different meals
calculate_tip(25.00, 20)
calculate_tip(45.50, 15)
```

```
Bill: $25.0
Tip: $5.0
Total: $30.0
Bill: $45.5
Tip: $6.825
Total: $52.325
```

The `return` Statement

Functions can send values back:

```
def calculate_tip_amount(bill, tip_percent):
    tip = bill * (tip_percent / 100)
    return tip

# Get the result and use it
lunch_tip = calculate_tip_amount(25.00, 20)
print(f"The tip is ${lunch_tip}")

# We can use the result in calculations
total = 25.00 + lunch_tip
```

return vs print

These are different!

```
def add_with_print(a, b):
    print(a + b)  # Shows on screen

def add_with_return(a, b):
    return a + b  # Sends value back

# Can't use print result
result1 = add_with_print(3, 4)  # Prints 7
print(result1)  # Prints None!

# Can use return result
result2 = add_with_return(3, 4)  # Returns 7
print(result2)  # Prints 7
```

```
7
None
7
```

Functions Can Return Early

```
def check_age(age):
    if age < 0:
        return "Invalid age!"
    if age < 18:
        return "Too young"
    return "Old enough"

print(check_age(-5))   # Invalid age!
print(check_age(16))   # Too young
print(check_age(21))   # Old enough
```

```
Invalid age!
Too young
Old enough
```

Once `return` runs, the function stops!

This works the same way that `break` works in loops!

Variable Scope

Variables inside functions are **local**:

```
def calculate():
    x = 10  # Local variable
    print(f"Inside function: x = {x}")

x = 5  # Different variable!
calculate()
print(f"Outside function: x = {x}")
```

```
Inside function: x = 10
Outside function: x = 5
```

But also, functions can access **global** variables:

```
x = 5  # Global variable
def calculate():
    print(f"Inside function: x = {x}")  # Uses global x
calculate()
print(f"Outside function: x = {x}")  # Also uses global x
```

```
Inside function: x = 5
Outside function: x = 5
```

Until now, all variables were global, but now we see the difference!

Building Bigger Programs

Functions let us break problems into pieces:

```
def get_bill_amount():
    return float(input("Enter bill amount: $"))

def get_tip_percent():
    return float(input("Enter tip percent: "))

def calculate_tip(bill, percent):
    return bill * (percent / 100)

def display_result(bill, tip):
    total = bill + tip
    print(f"\nBill: ${bill}")
    print(f"Tip: ${tip}")
    print(f"Total: ${total}")

# Main program
bill = get_bill_amount()
percent = get_tip_percent()
tip = calculate_tip(bill, percent)
display_result(bill, tip)
```

```
Enter bill amount: $
```

Common Function Patterns

```
# Validation function, returns True/False
def is_valid_grade(score):
    return score >= 0 and score <= 100

# Conversion function, converts input
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

# Menu function, displays options
def show_menu():
    print("1. Add")
    print("2. Subtract")
    print("3. Quit")
    return input("Choose: ")

# Main function, runs the program
def main():
    show_menu()
    choice = input("Enter your choice: ")
    if choice == '1':
        print("You chose Add")
```

Order of Functions

Functions can be defined in any order, but they must be defined for you to call them!

```
def greet():
    print("Hello before main!")

def main():
    greet()

main()
```

```
Hello before main!
```

```
def main():
    greet()

def greet():
    print("Hello after main!")

main()
```

```
Hello after main!
```

Functions Make Code Readable

Without functions:

```
print((float(input("Temperature: ")) * 9/5) + 32)
```

With functions:

```
def get_temperature():
    return float(input("Temperature in C: "))

def convert_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

temp_c = get_temperature()
temp_f = convert_to_fahrenheit(temp_c)
print(f"{temp_c}°C = {temp_f}°F")
```

Examples from Before

A function that asks if the user wants to continue, returns True/False

With a loop:

```
def should_continue():
    while True:
        answer = input("Continue? (yes/no): ")
        if answer in ['yes', 'no']:
            return answer == 'yes'
        print("Please enter 'yes' or 'no'.")
```

With recursion:

```
def should_continue():
    answer = input("Continue? (yes/no): ")
    if answer in ['yes', 'no']:
        return answer == 'yes'
    print("Please enter 'yes' or 'no'.")
    return should_continue()
```

Recursion

When a function calls itself, we call this **recursion**:

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)

print(factorial(5))
```

```
120
```

In some languages, there is no `while` or `for` loop, so recursion is the only way to repeat code!

If you can solve it with a loop, you can also solve it with recursion.

Function Best Practices

1. Give functions clear names that say what they do
   - Good: `calculate_average()`, `is_valid_password()`
   - Bad: `func1()`, `do_stuff()`
2. Keep functions focused on one task
3. Use parameters instead of relying on external variables
4. Return values when you need to use the result
5. Add comments to explain complex logic

Exercise: Temperature Converter

bigd103.link/temp-converter