Functions (Basics)

Key Concepts:

- Purpose of Functions
- Syntax of Functions
- Variable Scope

Purpose of Functions

Purpose: To structure your code when performing the same (or nearly the same) tasks multiple times.

- To make the code more readable (divide it into several parts)
- To make the code more easily modifiable
- To make the code easier to test

Example:

```
import turtle

def square(side_length):
    # Draw a square with the given side length
    i = 1  # side counter
    while i <= 4:
        turtle.forward(side_length)
        turtle.right(90)
        i += 1

# Main program
square(100)
turtle.up()
turtle.forward(130)
turtle.down()
square(50)
```

Principles

- A sequence of instructions encapsulated in a "box"
- Takes zero, one, or multiple arguments

- Returns zero, one, or multiple return values
- May have "side effects" that modify the environment (e.g., input/output interactions, turtle graphics)

Example: Calculating Distance

```
def distance(x1, y1, x2, y2):
    d = ((x2 - x1)**2 + (y2 - y1)**2)**0.5
    return d

if __name__ == "__main__":
    print(distance(1, 2, 1, 5))
    xA = 2
    yA = 3
    z = distance(xA, yA, 0, 0)
    print("Distance from (0,0) to A:", z)
```

Function Syntax

```
def function_name(argument1, ..., argumentN):
    # instructions to execute
    return return_value
```

Note: The `return` statement is optional, as are the arguments (but the parentheses are not).

Main Program

Place below the function definitions:

```
if __name__ == "__main__":
    # instructions to execute
```

Example:

```
if __name__ == "__main__":
    result = distance(2, 3, 4, 5)
    print(result)
    square(50)  # no return value, but has side effects (turtle)
```

Calling a Function from Another Function

Example:

```
def move_without_drawing(distance):
    turtle.up()
    turtle.forward(distance)
    turtle.down()

def line_of_squares(num_squares, side_length):
    i = 0
    while i < num_squares:
        square(side_length)  # call the square function
        move_without_drawing(side_length + 10)  # call another
function
        i += 1
```

Function without Arguments

Example:

```
import turtle

def standard_square():
    i = 1
    while i <= 4:
        turtle.forward(100)
        turtle.right(90)
        i += 1

def ask_name():
    name = input("What is your name?")
    return name

name = ask_name()  # no arguments
standard_square()  # no arguments and no return value
```

Side Effects vs. Return Values

Example with Return Values:

```
def add(x, y):
    return x + y
```

Example with Side Effects:

```
def add_IO():
    x = float(input("x? "))
    y = float(input("y? "))
    print(x + y)
```

Variable Scope

Each function has its own "set" of variables it can access. A variable:

- Created or modified in a function body
- Or that contains a function argument

is said to be local and is not accessible from the main program or another function.

Example:

```
def average(x, y):
    result = (x + y) / 2
    return result

if __name__ == "__main__":
    a = 5
    b = 6
    m = average(a, b)
    print(m)  # prints 5.5
    print(result)  # causes an error: NameError: name 'result' is not
defined
```

A variable defined in the main program cannot be modified inside a function (unless using the `global` keyword, which we will avoid).

Example:

```
def average(x, y):
    result = (x + y) / 2
    test = result  # creates a new variable test
    return result

if __name__ == "__main__":
    a = 5
    b = 6
    test = 0
    m = average(a, b)
    print("m =", m)  # prints "m=5.5"
    print("test =", test)  # prints "test=0"
```

Variables defined in the main program are accessible in read-only mode inside a function, but this is dangerous. We prefer passing all necessary values as arguments.

Correct Example:

```
def add_prefix(s, n):
    spaces = "-" * n
    result = spaces + s
    return result

if __name__ == "__main__":
    n = 5
    test = add_prefix("hello", n)
    print(test)
    n = 10
    test = add_prefix("world", n)
    print(test)
```

Summary:

- A variable created or modified in a function is local (exists only within the function).
- A variable from the main program cannot be modified inside a function.
- Pass all necessary values as function arguments.