LESSON 3: FUNCTIONS

FUNCTIONS -- LIKE VARIABLES FOR CODE

Functions are named blocks of code that are useful in 3 ways:

1. Organization & Reusability
2. Genericization of code
3. Interfacing between scripts

```
func transition(nextState:int):
    #transition to idle state
    if nextState == states.Idle:
        play("Idle")
        speed = 0

    #transition to walking state
    elif nextState == states.Walk:
        play("Walk")
        speed = maxSpeed

    #transition to attacking state
    elif nextState == states.Attack:
        play("Attack Straight")
        speed = 0

    elif nextState == states.Jump:
        play("Jump")
        speed = jumpSpeed

    currentState = nextState

func processState():
    #Process Idle state
    if currentState == states.Idle:
        #If a movement key is pressed, transition to walking state
        if checkMovementInput():
            transition(states.Walk)
        #If attack key is pressed, transition to attacking state
        elif checkAttackInput():
            transition(states.Attack)
```

Functions

2 PARTS OF A FUNCTION

To Utilize functions, you must do two things:

1. Define the function

   This is the majority of the work--Creating the code block, naming and formatting it.

2. Call the function

   Basically, telling the function to run from another place in your code.

```
func transition(nextState:int):
    #transition to idle state
    if nextState == states.Idle:
        play("Idle")
        speed = 0
```

```
func _ready():
    sprite = $"../Sprite"
    transition(states.Idle)
```

1. PROCESS OF DEFINING A FUNCTION

1. Functions can't be defined within other functions.
2. The function header must have at least this structure:
3. Optionally, you can declare arguments and a return type

```
func _ready():
    func myCoolFunction():
        pass
    pass # Replace with function body.
func myCoolFunction():
    pass
```

```
func name():
```

Return type

```
func name(thing1, thing2:int) -> int
```

Arguments

2. PROCESS OF CALLING A FUNCTION

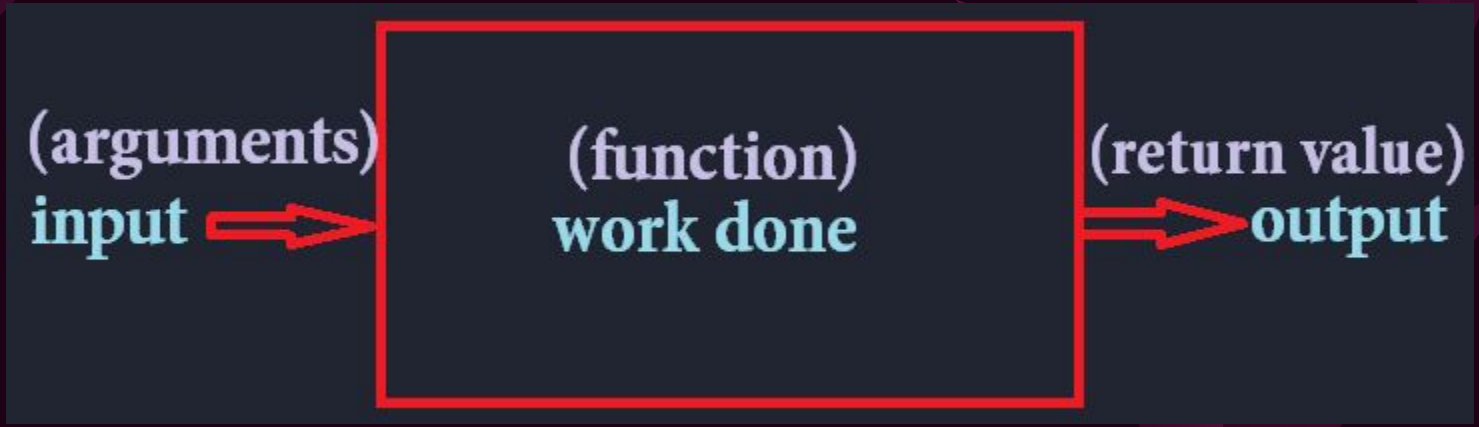
1. Functions must be called from within other functions.
2. If arguments were defined, provide them in the parentheses.
3. If a return type was defined, save the result of the func into a variable.

```
9  func _ready():
10     myCoolFunction()
11     pass
12
13 myCoolFunction()
```

```
myCoolFunction(1, myVar)
```

```
var result = myCoolFunction()
```

ARGUMENTS AND RETURN VALUE



Arguments allow you to pass data into a function. The function can use that data in its block of code, which can produce different outcomes.

Return values allow you to send the results of a function, or a message about the function, back to the spot it was called.

FUNCTION DEMO

CONTROL FLOW

```
9  func _ready():
10 
11     print("wawawawawawawa")
12 
13     myCoolFunction()
14 
15     print("uwuwuwuwuwuwuw")
16 
17 
18 func myCoolFunction():
19 
20     print("the quick brown fox")
21 
22     print("jumps over the lazy dog")
```

Called by game engine

Console Log:

CASE STUDY: INPUT FUNCTIONS

```
func is_pressed(key:String) -> bool
```

Is_pressed is a function defined by the engine. It returns true if a key is pressed, and false if a key is not pressed. But we don't know how the code inside works -- All we need to know is how to use it.

Argument allows user to 'control' the function by passing in a piece of data that represents the key that we want to check. 1 func can be used for any key we might want to check.

Return type sends us a signal back relating to the work done in the internals of the function. We don't need to know how it works, just that it will send back "true" or "false" based on the work done inside.

PRACTICE ASSIGNMENT

1. Create a completely new project. Import a sprite into the project.
2. Add a Sprite node to the scene. Assign an image to the sprite node.
3. Attach a new script to the Sprite node.
5. Define four functions named “moveUp” “moveDown” “moveLeft” and “moveRight”. No args or return values.
6. In each of these functions, write code to move your sprite in the appropriate direction.

   (Hint: position += Vector2(1, 0))
7. Inside of the _process() function, write an if-else ladder with 4 sections that check is_just_pressed() for your 4 arrow keys.
8. Inside of each section of this if-else ladder, call the corresponding move function

BONUS ROUND: Abstract the 4 move functions into 1 move function that takes a Vector2 as an argument.