CS105 Problem Solving

Functions

Wholeness

- Today we are going to look at functions, which let us take a part of our program and cleanly and clearly make it “it’s own thing”
- By encapsulating code in a function its variables become separate and can no longer be interfered with.

- Unity in Diversity – the same steps, from any location.

Functions

- We often have parts of code that 'do' a certain thing. If we want to do that thing in multiple places, we currently have to write it multiple times.
- Functions let you encapsulate a set of actions so that you can call on them from any place.
- In their encapsulation they also have their own variables, that do not interfere with the variables that you may have already defined.

Main Function / Other Functions

- So far we've only put steps into the main function

- It's possible – and actually advisable to also create other functions
  - Make the program more modular
  - Keeps everything short, clean, and clear

Example

Function for Turtle Graphics: square()

- Demo making the function
- Demo calling the funciton

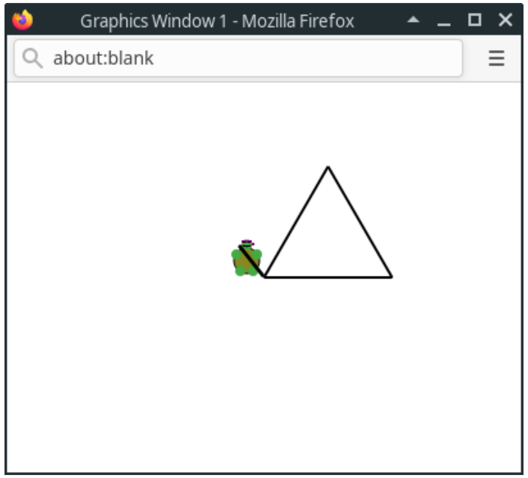
```
function triangle(turtle, size) {
    turtle.forward(size).rotate(-90);
    turtle.forward(size).rotate(-90);
    turtle.forward(size).rotate(-90);
    turtle.forward(size).rotate(-90);
    return 0;
}
```

Exercise

Write a turtle graphics function called triangle that takes a turtle and a size and creates an equilateral triangle whose sides are size.

Use it to create a triangle which sides are 100 as shown below:

- Once you have it working use it to create 3 different size triangles



Main Point 1

- Functions are a set of instructions (a flowchart) which can be called upon from your flowchart. By using functions we can organize our code so that each part of it (each function) is short and clear.
- Purification leads to progress

Parameters & Return

- The parameters area lets you define what type of data the function needs to receive in order to do its work.
- In our previous examples we saw that square() and triangle() both needed a turtle object.
- We didn't really pay attention to the return

Example

- We could also create function called myAdd that takes two numbers and returns the result of adding them.

Arguments

- The word argument is sometimes used interchangeably with the word parameter

- Officially:
  - Parameters are what a function will receive (variables, could be any value at code time)
  - Arguments are what the caller gives
    (values, given at run time)

Difference Between Parameters and Normal Variables

- Parameters are variables that receive a value from outside the function (from the arguments)

- Normal variables will not have any value until one is given to them inside the function

Functions Return Values

- Functions should not use output
  - Instead they should return

- This is a general design principle
  - All user interaction (input / output) should be kept in the main function
  - A function takes values, and then also gives a value back
    - The caller can then use the returned value to do additional things

Exercise

- Write a function called mySubt() that takes two numbers and returns the result of subtracting the second from the first

Main Point 2

- Functions should generally not use input and output. Instead they receive parameters (incoming values) and use their return to give the result of their computation to the caller (output).
- Every action has a reaction. The correct reaction to incoming parameters is to return a result – not output it!.

Summary

- Functions are a collection of instructions to achieve a task
- Functions can take parameters and have a return type
- Functions should not use output (instead return a value)