Subprograms

Functions

What is a Subprogram?

Most useful programs are larger than those we have considered so far. Unfortunately, as programs become larger, they also become more difficult to manage.

To address this issue, programmers break large programs into modules called subprograms.

This is also a useful way to implement the “divide and conquer” style of programming, by breaking large problems into smaller ones.

Functions

A function is a subprogram that returns a value of a particular data type (e.g., integer, real, string, boolean).

Turing provides many functions as a built-in part of the language.

For example, the square root function:

```
var answer : real

answer := sqrt(4)
% answer is now equal to 2
```

Example: Square Root Function

```
var answer : real

answer := sqrt(4)
% answer is now equal to 2
```

In general, we refer to this function as sqrt(x), where x is any real number. Since the square root of a real number is also real, the square root function returns a real value.

Look up this function in the Turing help <F10> and see how this information is presented.

sqrt **square root function**

Syntax

sqrt (*r* : **real**) : **real**

Description

The **sqrt** function is used to find the square root of a number. For example, **sqrt** (4) is 2.

Example

This program prints out the square roots of 1, 2, 3, … up to 100.

```
for i : 1 .. 100
    put "Square root of ", i, " is ", sqrt ( i )
end for
```

Details

It is illegal to try to take the square root of a negative number. The result of **sqrt** is always positive or zero. The opposite of a square root is the square. For example, the square of *x* is written is *x***2.

See also

See also predefined unit **Math**.

Other Examples of Functions

Function	Description	Returns
round()	rounds a real value	integer
length()	determines the length of a string	integer
sqrt()	calculates the square root of a real number	real
intstr()	converts an integer value to a string	string

Creating a Function

A function must be declared before it is used, similar to a variable. Thus the code for the function(s) should be placed at the top of your program.

For example:

```
function circumference(radius : real) : real
   % returns circumference of a circle
   result 2 * 3.14 * radius
end circumference

% output C for a circle with radius 10
put circumference(10)
```

Exercise: Function “circleArea”

Starting with the circumference code, add a second function to calculate the area of the circle. Ask the user for the radius and output the circumference and area for that circle.

Start with the circumference code:

```
function circumference(radius : real) : real
   % returns circumference of a circle
   result (2 * 3.14 * radius)
end circumference

% output C for a circle with radius 10
put circumference(10)
```

Exercise: Function “circleArea”

```
function circumference(radius : real) : real
   % returns circumference of a circle
   result 2 * 3.14 * radius
end circumference

function circleArea(radius : real) : real
   % returns area of a circle = pi * r-squared
   result 3.14 * radius * radius
end circleArea

var radius : real

% output C and A for a given radius
put "Enter the radius of your circle: "..
get r
put "C = ", circumference(radius)
put "A = ", circleArea(radius)
```

Functions – Input & Output

The function is like a machine that takes your input and produces some sort of output.

For example, there are “change machines” that will allow you to input a $5, 10, or 20 bill. In return, it will output change (usually $2, 1, or 0.25 coins).

Input/Output Data Types

The input to a “function machine” can involve different types of input or output.

A change machine accepts currency (bills). The output is also currency (coins).

A pop machine accepts currency (coins), but also requires the user to press some buttons (strings?) to decide on their purchase. The output is yet another type (beverage).

Input/Output Data Types

When programming, our data types (so far) are integer, real, string, and boolean.

A function can use any combination of these as input. The input can be one or more parameters, and they can be the same or different data types.

For output, the function will return a single value of one data type. It can be the same or different than any of the input data types.

Declaring a Function – Data Types

By declaring a function, we specify both the input and output data types:

```
function something ( x : type1, y : type2 ) : type3
```

In this case, the input x is data type #1 and the input y is data type #2. They are just like var statements, without the "var".

The function itself is data type #3. Notice this type is declared outside the bracket. This is the output data type.