



Introduction to Programming

Pass Task 1.2: My Functions

Overview

Procedures are a great tool for capturing the instructions needed to perform a task, but sometimes you need to be able to capture the instructions needed to calculate a value. Using functions you can now create artefacts to encapsulate the steps needed to calculate a value.

- Purpose:** Learn how to create your own functions.
- Task:** Use the following instructions to implement a function and use it to calculate a value based on user input.
- Time:** This task should be completed before the start of week 3.
- Resources:**
- Chapter 4 of the Programming Arcana
 - Swinburne CodeCasts ([YouTube Channel](#), [iTunesU](#))
 - [Creating your own Functions](#)
 - Syntax Videos
 - [Calling Functions](#), [Creating Your Own Functions](#)

Submission Details

You must submit the following files to Doubtfire:

- Program source code demonstrating your creation and use of functions
- Screenshot of the Terminal showing the execution of your program.

Make sure that your task has the following in your submission:

- The program must use one of your functions to calculate a return value.
- Code must follow the Pascal coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show it working.
- Your program must demonstrate the use of functions with parameters, as well as constants.

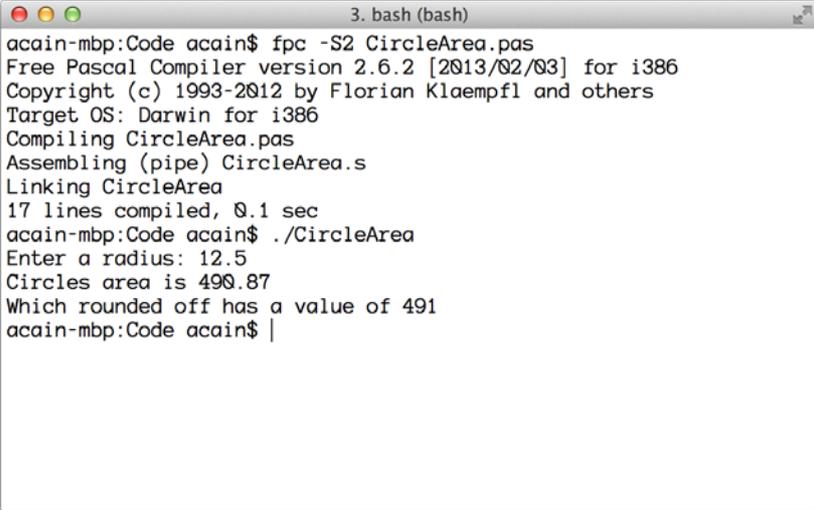
About Functions

In programming, functions are used to calculate a value. They are very similar to procedures, with the added feature that they **return** a **result** when they end. For example, the following program uses the [Pi](#), [Sqr](#) and [Round](#) functions to calculate the area of a circle.

```
program CircleArea;
uses TerminalUserInput;

procedure Main();
var
  cirRadius, cirArea: Single;
  roundedValue: Integer;
begin
  cirRadius := ReadInteger('Enter a radius: ');
  cirArea := Pi() * Sqr(cirRadius);
  WriteLn('Circles area is ', cirArea:4:2);
  roundedValue := Round(cirArea);
  WriteLn('Which rounded off has a value of ', roundedValue);
end;

begin
  Main();
end.
```



```
3. bash (bash)
acain-mbp:Code acain$ fpc -S2 CircleArea.pas
Free Pascal Compiler version 2.6.2 [2013/02/03] for i386
Copyright (c) 1993-2012 by Florian Klaempfl and others
Target OS: Darwin for i386
Compiling CircleArea.pas
Assembling (pipe) CircleArea.s
Linking CircleArea
17 lines compiled, 0.1 sec
acain-mbp:Code acain$ ./CircleArea
Enter a radius: 12.5
Circles area is 490.87
Which rounded off has a value of 491
acain-mbp:Code acain$ |
```

Tip: The Free Pascal website has a list of all of the [Mathematical functions](#) available to all Pascal programs, as well as a [list of units](#) that you can use if you want access to additional functionality.

In this program we could create a **CircleArea** function that calculates the area of a circle. That would mean that we could use that function any time we wanted to get the area of a circle.

Designing a function is just like designing a procedure. One way of approaching it is to think that someone has asked you to perform the calculation. Putting yourself in that position will help you think about the data the function will need to be given, and the data it will return. In this case you need to be told the radius of the circle, and you then return the area. The radius will be a number, and could have a fractional part so you use the **Single** data type. The result will then be a number and could have a fractional part, so it is also a Single. This gives you the *functions prototype* - which is all the details about the functions name, its parameters, and re- turn type.

The body of the function will then calculate the area of the circle, from the radius parameter. It will then set this as the result of the function. The value in the result variable is returned to the caller when the function ends.

```
function CircleArea(radius: Single): Single;  
var  
    area: Single;  
begin  
    area := Pi() * Sqr(radius);  
    result := area;  
end;
```

You can then call the CircleArea function in Main, and store the *result* of calling the function into Main's cirArea variable.

```
procedure Main();  
var  
    cirRadius, cirArea: Single;  
    roundedValue: Integer;  
begin  
    cirRadius := ReadInteger('Enter a radius: ');  
    cirArea := CircleArea(cirRadius);  
    WriteLn('Circle area is ', cirArea:4:2);  
    roundedValue := Round(area);  
    WriteLn('Which rounded off has a value of ', roundedValue);  
end;
```

Note: Pascal creates the **result variable** for you automatically from the function declaration. The following code highlights where result is "declared".

```
function CircleArea(radius: Single): Single;
```

Instructions

Functions allow you to build modular programs that take parameters and return a result.

To explore this topic, we will modify the Terminal **SimplePolitics.pas** program so that it will:

- Prompt the user to enter his or her name and birth year.
- Call a function to calculate how old the user was when Trump was elected President of the USA (you will need to write this function yourself).
- Print to the terminal the user's name and age when Trump was elected.
- Call the **ReadBoolean(prompt: String): Boolean**; function from TerminalUserInput.pas to prompt the user to enter whether he or she is a supporter of Brexit.

Note: You will need to use the `ReadBoolean()` function in the file `TerminalUserInput` (and add `uses TerminalUserInput` to the top of `SimplePolitics.pas`)

- Print out whether the user supports Brexit or not based on the result of calling the `ReadBoolean()` function.

Procedure: **Main**

Variables:

- Const YEAR_TRUMP_ELECTED = 2016 (to store an Integer value read from the file)
- yearBorn (to store String values read from the file)

Steps:

1. Prompt the user to enter his or her name.
2. Read in the user's name.
3. Prompt the user to enter his or her year of birth.
4. Read in the user's year of birth.
5. Call the function `calculateAgeWhenTrumpElected(birthYear: Integer): Integer`; to calculate how old the user was when Trump was elected President of the USA.
6. Print to the terminal the user's name and the message: 'you were X years old when Trump was elected'.
7. Call the function `ReadBoolean(prompt: String): Boolean`.
8. Print out the user's name, then either 'is a Brexit supporter' or 'is NOT a Brexit supporter' depending on the output of the function `ReadBoolean()`.
9. Request the user to "Press Enter to Continue".
10. Read a blank line.

- Compile and run your program.

Now that the Task is complete you can submit it for assessment, which will help prepare it for your portfolio.

1. Use [Skitch](#) (or your preferred screenshot program) to take a screenshot of the Terminal, as this is one of the things you will need to submit.
2. Save the document and backup your work to multiple locations!
 - Once you get things working you **do not** want to lose them.
 - Work on your computer's storage device most of the time... but backup your work when you finish each task.
 - Use **Dropbox** or a similar online storage provider, as well as other locations.
 - Doubtfire is not a Backup of your work, so make sure you keep a copy!
 - A USB key and portable hard drives are good secondary backups... but can be lost/damaged (do not rely upon them).
3. Login to Doubtfire, and locate Pass Task 1.2
4. Change the status of the task to **Ready To Mark**
5. Upload your completed Hello World code and the screenshot.
6. If you check back later Doubtfire will have prepared these as PDFs for your tutor to assess.

You now have another of your first portfolio pieces. This will help demonstrate your learning from the unit.

check **Note**: This is one of the tasks you need to **submit to Doubtfire**. Check the assessment criteria for the important aspects your tutor will mark.

End of Task