Lecture 3 - Converting Values

Overview

Also the lecture notes are online in the Lect-03 github. https://github.com/Univ-Wyo-Education/S22-1010/tree/main/class/lect/lect-03

There is a ./conv directory that has a series of steps where you can go back to this and see the code as I develop it.

The “Process” that we often use

1. Collect Requirements
2. Produce a “design” / or a Requirements Document
3. Convert Requirements into a Prototype
4. Take the Prototype and build test data from it
5. Build the “application” from the Prototype
6. Build automated tests
7. Document results

Demo - of this in browser.

A lot of what happens when you program seems so simple - until you have to learn a non-human language. Programs are formal languages. English is an informal language. For example I can make a sentence that most of you will not understand, at first, but with some explanation I can show that it is using proper English crammer.

“The old man the boat.”

In this context the old is a type of person. “man” is to get on board the boat and operate it. It is a verb.

So... The sentence is roughly equivalent to “The old people get on the boat and operate it.”

Python is a formal language. It uses a rigorous syntax. As humans we are not used to this.

One of the realities of development is that you are not using “one” tool. vim - for editing, or notepad++ on windows (don’t use notepad it will mess you up). VSCode - for debugging - and building and running projects. Python - command line for running programs.
Jupiter Notebooks (Iron Python) for mixing code and output in a human readable form. Different types of files. .py for python, .txt for text, .csv for comma separated values, .mk for markdown, images in .svg and .jpg and .png etc.

Software Engineering

This is a very “software engineering” approach to code development. Learning to code effectively is a “process”. Creativity tends to be innate - it is a talent. Programming is a set of skills.

Topics Covered

1. Files and Directories
2. Editing

3. `def` code reusability
4. Basic testing
5. Functions - parameters - return values
6. if
7. Comparison for equality, `==` operator. Also `!=` not equal.
8. if / else
9. ':' starts a block
10. Indentation
11. a = a + 1 - not algebra
12. Files
13. Patterns in code
14. Better testing

Requirements

Implement a python function that will convert from size of sun to the size of a tennis ball (2.75 inches) and return that value. Input should be in miles, output in inches.

Implement a program that will use the function, prompt for input in miles and then print out the result in kilometers.

Step 0

Break the problem down.

When we start with the sun in miles and the tennis ball in inches - we have to have a common unit.
To do this we need a conversion capability from miles into inches.

Step 1

Convert from miles to inches.

Conversion generally is ((X + k1) * C) + k2

In our case k1 and k2 are 0. So we just get X * C

Demo - lookup conversion from miles to kilometers

```
1: # Step 1 – constants
2:
3: miles = 3
4: conv = 5280 * 12
5: inch = miles * conv
6: print ( inch )
7:
```

Demo - of this as a visualization

Step 2 - Input with error

```
 1: # Step 2 - will error with type error
 2:
 3: print ( "Enter Miles" )
 4:
 5: miles = input()
 6:
 7: conv = 5280 * 12
 8: inch = miles * conv
 9:
10: print ( "inch = {}".format(inch) )
```

Step 3 - Fixed error / Types

```
 1: # Step 3 - inline after fixing type
 2:
 3: print ( "Enter Miles" )
 4:
 5: miles_str = input()
 6: miles = float(miles_str)
 7: conv = 5280 * 12
 8: inch = miles * conv
 9:
10: print ( "inch = {}".format(inch) )
```

Step 4 - Make a function

```
 1: # Step 4 - After making a function
 2:
 3: def mi_to_inch ( mi ):
 4:     conv = 5280 * 12
 5:     inch = mi * conv
 6:     return (inch)
 7:
 8: print ( "Enter Miles" )
 9:
10: miles_str = input()
11: miles = float(miles_str)
12:
13: inch = mi_to_inch(miles)
14:
15: print ( "inch = {}".format(inch) )
```

Step 5 - Make Reusable Code

step-5.py:

```
1: # Step 5 - with function and a test.
2:
3: import conv_mi_to_inch
```

```
 4:
 5: print ( "Enter Miles" )
 6:
 7: miles_str = input()
 8: miles = float(miles_str)
 9:
10: inch = conv_mi_to_inch.mi_to_inch(miles)
11:
12: print ( "inch = {}".format(inch) )
```

conv/mi_to_km.py:

```
 1: # mi_to_in converts from miles as an integer or float to inches.
 2: def mi_to_inch ( mi ):
 3:     conv = 5280 * 12
 4:     inch = mi * conv
 5:     return (inch)
 6:
 7: # Automated Test
 8: if __name__ == "__main__":
 9:     n_err = 0
10:     x = mi_to_inch ( 3 )
11:     if x !=  190080.0: # is equal to 5280 * 12 * 3
12:         n_err = n_err + 1
13:         print ( "Error: Test 1: conversion not working, expected {} got {}".
14:                 format (  4.82802, x ) )
15:     x = mi_to_inch ( 0 )
16:     if x != 0:
17:         n_err = n_err + 1
18:         print ( "Error: Test 2: conversion not working, expected {} got {}".
19:                 format ( 0, x ) )
20:
21:     if n_err == 0 :
22:         print ( "PASS" )
23:     else:
24:         print ( "FAILED" )
```

Step 6 - Add documentation

This is really a little step in this program - but a really important one for this class..

```
 1: # Author: Philip Schlump
 2: # Email: pschlump@uwyo.edu
 3:
 4: # Main program to read in values and convert from miles (mi) to kilometers (inch)
 5:
 6: # Step 6 - with function and a test.
 7:
 8: import conv_mi_to_inch
 9:
10: print ( "Enter Miles" )
11:
12: miles_str = input()
13: miles = float(miles_str)
14:
15: inch = conv_mi_to_inch.mi_to_inch(miles)
16:
```

```
17: print ( "inch = {}".format(inch) )
```

And our final version

```
 1: # Author: Philip Schlump
 2: # Email: pschlump@uwyo.edu
 3:
 4: # Main program to read in values and convert from miles (mi) to kilometers (inch)
 5:
 6: # Step 6 - with function and a test.
 7:
 8: import conv_mi_to_inch
 9:
10: print ( "Enter Miles" )
11:
12: miles_str = input()
13: miles = float(miles_str)
14:
15: inch = conv_mi_to_inch.mi_to_inch(miles)
16:
17: print ( "inch = {}".format(inch) )
```

Copyright