CptS 111, Spring 2023
Lect. #12, Feb. 27, 2023
Class Notes

Today's Agenda:

1. The `main()` function
2. Function stubs
3. Common errors

Ch. 5 (cont.)

1. The `main()` Function

Consider the following program for calculating someone's BMI:

In [1]:
```
# Program with two non-void functions and some statements that are run
# after the functions have been defined.

def get_input():
    weight = float(input('Enter weight [pounds]: '))
    height = float(input('Enter height [inches]: '))
    return weight, height

def calc_bmi(weight, height):
    bmi = 703 * weight / (height * height)
    return bmi

wt, ht = get_input()                # func call w/ 2 lvalues
bmi = calc_bmi(wt, ht)              # func call w/ 1 lvalue
print(f'Your BMI is {bmi:.0f}.')
```

```
Enter weight [pounds]: 165
Enter height [inches]: 72
Your BMI is 22.
```

In this program we first defined two non-void functions. These two functions are called in the statements written below the function definitions. **In Python it's good practice to collect all these statements into another function called `main()`.** To run the entire program, we issue the simple function call `main()`

In [2]:
```
# Program with two non-void functions that are called by the void functio
# main().  To run the program, we issue a simple call to main().

def get_input():
    weight = float(input('Enter weight [pounds]: '))
    height = float(input('Enter height [inches]: '))
    return weight, height

def calc_bmi(weight, height):
    bmi = 703 * weight / (height * height)
    return bmi

def main():
    wt, ht = get_input()
    bmi = calc_bmi(wt, ht)
    print(f'Your BMI is {bmi:.0f}.')

main()
```

```
Enter weight [pounds]: 165
Enter height [inches]: 72
Your BMI is 22.
```

Note that `main()` never has any parameters, and it's the only function for which we *should not* use a `return` statement.

From now on we'll always include a `main()` function in all our programs. Think of `main()` as the place to start if we want to figure out what a program is doing, i.e., we look at the main part of the program! We can also use it to organize our code.

Let's consider another example.

In [3]:
```
# Program to calculate payments for a student loan.
# Notice the use of the commas in the replacement fields {}

def get_input():
    '''
    Non-void function that prompts user for loan values: loan amt, intere
    number of years to pay off loan
    '''
    loan_amt = float(input('Enter loan amount: '))
    percent_int = float(input('Enter interest rate [%]: '))
    num_years = int(input('Enter number of years to pay off loan: '))
    int_rate = percent_int / 100
    num_months = num_years * 12
    return loan_amt, int_rate, num_months

def calc_pymts(loan, int_rate, n):
    '''
    Non-void function that calculates loan values: monthly pymt, total pa
    total interest paid
    '''
    num = loan * int_rate / 12
    denom = 1 - (1 + int_rate / 12) ** -n
    month_pymt = num / denom
    pymt_total = month_pymt * n
    int_total = pymt_total - loan
    return month_pymt, pymt_total, int_total

def print_pymts(month, total, interest):
    '''
    Void function that prints loan values: monthly pymt, total paid, tota
    '''
    print()  # line to separate input and output
    print(f'Your monthly payment will be ${month:,.2f}.')
    print(f'The total amount you will end up paying is ${total:,.2f}.')
    print(f'The total amount of interest you will pay is ${interest:,.2f}
    return

def main():
    loan, int_rate, num_months = get_input()
    month_pymt, pymt_total, int_total = calc_pymts(loan, int_rate, num_mo
    print_pymts(month_pymt, pymt_total, int_total)

main()
```

```
Enter loan amount: 10000
Enter interest rate [%]: 7
Enter number of years to pay off loan: 10

Your monthly payment will be $116.11.
The total amount you will end up paying is $13,933.02.
The total amount of interest you will pay is $3,933.02.
```

What's good about this code is that you can look at the commands in `main()` and tell very quickly what the program is doing. To polish off this program, you need to add a header to the program, docstrings, and comments where appropriate. If a function isn't

working correctly, you simply need to fix it without worrying about the rest of the code.

2. Function Stubs

In the program we just discussed, we wrote four functions including `main()`. To practice good programming, it's best to program in increments, checking to make sure the code you've written does what you expect it to do. To do this, we usually create "skeletons" of the functions we're going to code, and then we write our `main()` function. Once we know that it's working correctly, we write one function, check to make sure everything works, and then continue to the next function. In the skeleton code for our functions, we can use function stubs. In CptS 111, we'll use the function stub `pass`. We can also use a print statement, but this requires that we add a `return` value if we're working with a non-void function.

Suppose we want to write a program to calculate the BMI of someone using three functions: `get_input()`, `calc_bmi()`, and `main()`. We can quickly write the following:

```
def get_input():
    pass

def calc_bmi():
    pass

def main():
```

Then we can think about what steps we need to calculate the BMI and fill them in in `main()`:

```
def main():
    wt, ht = get_input()
    bmi = calc_bmi(wt, ht)
    print(f'Your BMI is {bmi:.0f}.')
```

Before proceeding to write the other functions, we make sure `main()` works correctly first by commenting out the function calls and entering a fixed value for the `bmi` which we'll delete after making sure `main()` works:

In [4]:
```
def get_input():
    pass

def calc_bmi(weight, height):
    pass

def main():
#     wt, ht = get_input()
#     bmi = calc_bmi(wt, ht)
    bmi = 20.589341
    print(f'Your BMI is {bmi:.0f}.')

main()
```

```
Your BMI is 21.
```

Of course, here `main()` is really simple so we don't really need to test it, but if it were more complex, we would want to do so. After determining that `main()` is correct, we then move on to the `get_input()` function:

In [5]:
```
def get_input():
    print('***Made it to get_input()***')  # print() is useful for isolat
    weight = float(input('Enter weight [pounds]: '))
    height = float(input('Enter height [inches]: '))
    return weight, height

def calc_bmi(weight, height):
    pass

def main():
    wt, ht = get_input()
    print(f'wt and ht are {wt:g} and {ht:g}.')
#    bmi = calc_bmi(wt, ht)
#    bmi = 20.589341
#    print(f'Your BMI is {bmi:.0f}.')

main()
```

```
***Made it to get_input()***
Enter weight [pounds]: 165
Enter height [inches]: 72
wt and ht are 165 and 72.
```

Notice how we can use the `print()` function to ensure the values returned by `get_input()` are what we expected. **I often use `print()` to try to track down errors in my code! You'll find this to be useful in PA #6 and PA #7.** Next, we move on to the `calc_bmi()` function, which we completed previously above. However, note that deconstructing programs into steps makes it easier to keep track of what we're doing, and it's a good way to isolate any errors in our logic!

3. Common Programming and Function Errors

1. Syntax errors: forget a parenthesis, indent incorrectly, etc.; program won't run
2. Run-time errors: wrong type of input, variable undefined, and so on
3. Logic errors: most difficult to find; program runs but wrong value returned.
4. Function errors: forget to change variable names when copying and pasting code; forget to return a value

For logic errors, it's a good idea to use the `print()` function to print out values. This allows us to do two things: 1) figure out which values are incorrect and 2) figure out where our errors are occurring, e.g., sometimes a function never gets called.

```
def calc_bmi(weight, height):
    print('Values of weight and height are:', weight, height)
    <body>
    return
```

In future programming courses, you'll learn how to use debuggers.