Another Problem

- Tip Calculator
  - A trivial example of a calculation
  - Better implementation: show me the tip amounts for 12%, 15%, 18% and 20% all at once

- How do we automate this?
  - Identify the data, known and unknown
  - Describe the interaction with the machine in detail (the appropriate level of abstraction)
  - Work out the relationships between the data (equations)
  - Write the algorithm for the machine to follow

A General-Purpose Machine

- Programmable machines (computers) rule the world!
- Program
  - Simplified, a sequence of *instructions*
  - Each instruction tells the computer to perform an *operation*
  - Levels of abstraction
    - *Machine code*: instructions that the hardware can actually perform
      - □ Arithmetic, storing and moving numbers
    - *Assembly language*: human readable machine code
    - *High-level languages*: layers of abstraction create instructions that represent multiple instructions at the machine level
      - □ E.g. "go buy a car" vs. "open the door, take a step, etc"
  - Many, many high-level languages, with different abstractions
    - C > C++ > Java/C# > Scripting Languages (JavaScript, Python)

Python Programming

- Python 3
  - One of the "scripting languages"
  - Higher level of abstraction
  - Closer to the problem, more productive
  - Less control, less performance
  - Easier to abstract away details that **do matter** and get the solution wrong!

Tools

- A text editor to write the code
  - Code goes in plain old text files, extension .py by convention
  - IDLE to start (comes with Python)
  - Notepad++, Atom, Visual Studio Code, Sublime Text…
- An *interpreter* to run code
  - The interpreter turns Python instructions into machine-level instructions as the program runs
  - Contrast: *compiled* languages like C++ convert to machine code **before** you run the program
    - More performance, less flexibility
- Python code can be done interactively (one line at a time) or by running whole files
  - The former is just for testing, exploring

Back to our problem

- We need…A way to interact!
  - Command line first
  - Print to the screen, read what the user types back

Back to our problem

- A bit more detail (moving to a lower level of abstraction)
  - The computer can perform certain actions
  - How do we command it to do so?
    - By naming them! (sensible, right?)
    - Simon says…

Back to our problem

- A bit more detail (moving to a lower level of abstraction)
  - The computer can perform certain actions
  - How do we command it to do so?
    - By naming them! (sensible, right?)
    - Simon says…
  - Some commands need more information
    - Stand up does not
    - Jump up and down does not
    - Raise your hand does
    - Write does
    - We call that extra information *parameters*

Back to our problem

- The actions the computer can perform are called *functions*
  - We *call* a function to make the computer perform it

```
jump()
```

  - Okay, it doesn't know how to jump

```
print()
```

  - Print needs more information!

```
print(17)
print("This is a string")
```

    - Specifically, it requires a piece of data to print
    - Numbers are data, so are *strings* of characters
      - □ Quotes tell Python to treat the characters as string data

Back to our problem

- We need…A way to store and manipulate data!
  - Variables
    - Named "boxes" that hold pieces of information (data)
    - An *assignment statement* tells Python to create a variable, name it, and put a *value* in it

```
name = "Tom"
```

    - Variables can be used anywhere you can use data

```
print(name)
```

    - This tells the computer to print the value in that variable
      - □ Implied: go get the value from the variable, then pass it into the function

Back to our problem

- We need…A way to store and manipulate data!
  - In addition to functions, we also use *operators*
    - Just a different syntax, that looks like familiar math equations
    - Each operator takes two arguments (one on the left, one on the right)
  - The assignment operator (=)

```
age = 17
```

- Puts a value into a variable, creating it if necessary
- Not the same as mathematical equality!

```
age = 8
age = 9
```

- Makes no sense in math, here means to assign, then reassign (overwrite)

Back to our problem

- We need…A way to store and manipulate data!
  - Arithmetic operators (+, -, *, /, //, %)
    - The *expression* 6 + 7 *evaluates* to 13
    - The *expression* 14 / 4 *evaluates* to 3.5
  - Expressions can be chained together
    - 6 + 8 + 2 + 9 evaluates to 25
    - Tells the computer to add 6 and 8 (evaluates to 14)
    - Then add 14 and 2 (evaluates to 16)
    - Then add 16 and 9 (evaluates to 25)
  - Follows standard arithmetic order of operations
    - Multiplication and division first, parenthesis to force grouping

Back to our problem

- We need…A way to get user input!
  - Still on the command line
    - The function `input()`
    - Tells the program to wait for the user to type, then give us the characters that the user typed
  - Functions can take in data (parameters)
  - Functions can also *return* data
    - Just like 2 + 3 evaluates to 5
    - input() evaluates to whatever the user typed
  - Store the result of input() just like any expression

```
number = 2 + 3
name = input()
```