"I was able to work with a team of Peruvian faculty on building and coding a rapid response ventilator system for patients with COVID-19. After weeks of hard work and tinkering, our team achieved excellent results in tests by doctors and now we got approval from the Peruvian health ministry to move on with the production of our first thousand ventilators."

Marcelo Pena
Freshman Computer Science major at Stanford University

Martin Vizcarra
President of Peru
def main():
    print("This program adds two numbers.")
    num1 = input("Enter first number: ")
    num1 = int(num1)
    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
    print("The total is " + str(total) + ".")
def main():
    print("This program adds two numbers.")
    num1 = int(input("Enter first number: "))

    num2 = input("Enter second number: ")
    num2 = int(num2)
    total = num1 + num2
    print("The total is " + str(total) + ".")
def main():
    print("This program adds two numbers.")
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    total = num1 + num2
    print("The total is " + str(total) + ".")
def main():
    print("This program adds two numbers.")
    num1 = int(input("Enter first number: "))
    num2 = int(input("Enter second number: "))
    total = num1 + num2
    print("The total is " + str(total) + ".")

• Often, this is how you'll see code that gets input
• But, what if I want to do more than add?
• It's time for the world of *expressions*
1. Understanding arithmetic expressions
2. Using constants
3. Random number generation
Arithmetic Operators

num1 = 5
num2 = 2

• Operations on numerical types (*int and float*)

• Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>&quot;addition&quot;</td>
<td>num3 = num1 + num2</td>
<td>7</td>
</tr>
<tr>
<td>-</td>
<td>&quot;subtraction&quot;</td>
<td>num3 = num1 - num2</td>
<td>3</td>
</tr>
<tr>
<td>*</td>
<td>&quot;multiplication&quot;</td>
<td>num3 = num1 * num2</td>
<td>10</td>
</tr>
<tr>
<td>/</td>
<td>&quot;division&quot;</td>
<td>num3 = num1 / num2</td>
<td>2.5</td>
</tr>
<tr>
<td>//</td>
<td>&quot;integer division&quot;</td>
<td>num3 = num1 // num2</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>&quot;remainder&quot;</td>
<td>num3 = num1 % num2</td>
<td>1</td>
</tr>
<tr>
<td>**</td>
<td>&quot;exponentiation&quot;</td>
<td>num3 = num1 ** num2</td>
<td>25</td>
</tr>
<tr>
<td>-</td>
<td>&quot;negation&quot; (unary)</td>
<td>num3 = -num1</td>
<td>-5</td>
</tr>
</tbody>
</table>
Precedence

• Precedence of operators (in order)
  
  ( ) "parentheses" highest
  
  ** "exponentiation"

  - "negation" (unary)

  *, /, //, %

  +, - lowest

• Operators in same precedence category are evaluated left to right

  – Similar to rules of evaluating expressions in algebra
Precedence Example

\[ x = 1 + 3 \times 5 / 2 \]

\[ \text{Tree Structure:} \]

1. \[ 15 \]
2. \[ 7.5 \]
3. \[ 8.5 \]
4. \[ x = 8.5 \]
Implicit Type Conversion

num1 = 5
num2 = 2
num3 = 1.9

• Operations on two ints (except /) that would result in an integer value are of type int
  num1 + 7 = 12  \text{(int)}
  – Dividing (/) two ints results in a float, even if result is a round number (Ex.: \(6 / 2 = 3.0\))

• If either (or both) of operands are float, the result is a float
  num3 + 1 = 2.9  \text{(float)}

• Exponentiation depends on the result:
  num2 ** 3 = 8  \text{(int)}
  2 ** -1 = 0.5  \text{(float)}
Explicit Type Conversion

num1 = 5
num2 = 2
num3 = 1.9

- Use \texttt{float(value)} to create new real-valued number
  \[
  \text{float}(\text{num1}) = 5.0 \quad \text{(float)}
  \]
  - Note that \texttt{num1} is not changed. We created a new value.
  \[
  \text{num1} + \text{float}(\text{num2}) = 7.0 \quad \text{(float)}
  \]
  \[
  \text{num1} + \text{num2} = 7 \quad \text{(int)}
  \]

- Use \texttt{int(value)} to create a new integer-valued number (truncating anything after decimal)
  \[
  \text{int}(\text{num3}) = 1 \quad \text{(int)}
  \]
  \[
  \text{int}(-2.7) = -2 \quad \text{(int)}
  \]
Float is Not Always Exact

num1 = 5
num2 = 2
num3 = 1.9

• What is type of: num3 - 1
  – Answer: float

• What is value of: num3 - 1
  – Answer: 0.8999999999999999
  – WHAT?!

I find your lack of precision disturbing!

Don't be so negative, Darth Integer!
Expression Shorthands

| num1 = 5    |
| num2 = 2    |
| num3 = 1.9  |

num1 = num1 + 1    same as    num1 += 1
num2 = num2 - 4    same as    num2 -= 4
num3 = num3 * 2    same as    num3 *= 2
num1 = num1 / 2    same as    num1 /= 2

• Generally:
  \( variable = variable \) operator \( (expression) \)
  is same as:
  \( variable \) operator\( = expression \)
Let's consider an example `average2numbers.py`
average2numbers.py

"""
File: average2numbers.py
------------------------
This program asks the user for two numbers and prints their average.
"""

def main():
    print("This program averages two numbers.")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    total = (num1 + num2) / 2
    print("The average is " + str(total) + ".")

    # This provided line is required at the end of a Python file to call the main() function.
if __name__ == '__main__':
    main()
Constants

- Constants make code easier to read (good style):
  \[
  \text{area} = \text{PI} \times (\text{radius} \times \times 2)
  \]
  - Written in all capital SNAKE_CASE with descriptive names
  - Constant are really variables that represent quantities that don’t change while the program is running
  - Can be changed between runs (as necessary)
    - "Hey, we need to compute a trajectory to get us to Mars"
  - Code should be written with constants in a **general** way so that it still works when constants are changed

INCHES_IN_FOOT = 12
PI = 3.1415
Example of Using Constants

File: constants.py
------------------
An example program with constants

```
INCHES_IN FOOT = 12

def main():
    feet = float(input("Enter number of feet: "))
    inches = feet * INCHES_IN FOOT
    print("That is " + str(inches) + " inches!"

# This provided line is required at the end of a Python file
# to call the main() function.
if __name__ == '__main__':
    main()
```

Piech and Sahami, CS106A, Stanford University
import math

• math library has many built-in constants:
  math.pi  mathematical constant \( \pi \)
  math.e  mathematical constant \( e \)

• and useful functions:
  math.sqrt(x)  returns square root of \( x \)
  math.exp(x)  returns \( e^x \)
  math.log(x)  returns natural log (base \( e \)) of \( x \)

• These are just a few examples of what's in math
Example of Using \texttt{math} Library

\begin{verbatim}

"""
File: squareroot.py
-------------------
This program computes square roots
"""

\texttt{import} \texttt{math}

\texttt{def} \texttt{main}():
    \texttt{num} = \texttt{float}(...)\texttt{input("Enter number: ")}
    \texttt{root} = \texttt{math.sqrt}(...)\texttt{num})
    \texttt{print("Square root of ", \texttt{str}(\texttt{num}) + " is ", \texttt{str}(\texttt{root}))}

# This provided line is required at the end of a Python file
# to call the \texttt{main}() function.
\texttt{if} \texttt{\_\_name\_\_} == \texttt{\'\_\_main\_\_\'):
    \texttt{main()}
\end{verbatim}
Random Number Generation

• Want a way to generate random number
  – Say, for games or other applications
• No "true" randomness in computer, so we have *pseudorandom* numbers
  – "That looks pretty random to me"
• Want "black box" that we can ask for random numbers

Next random number? 5

Next random number? 3

• Can "seed" the random number generator to always produce the same sequence of "random" numbers
## Python random Library

```python
import random
```

<table>
<thead>
<tr>
<th>Function</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>random.randint(min, max)</code></td>
<td>Returns a random integer between <code>min</code> and <code>max</code>, inclusive.</td>
</tr>
<tr>
<td><code>random.random()</code></td>
<td>Returns a random real number (float) between 0 and 1.</td>
</tr>
<tr>
<td><code>random.uniform(min, max)</code></td>
<td>Returns a random real number (float) between <code>min</code> and <code>max</code>.</td>
</tr>
<tr>
<td><code>random.seed(x)</code></td>
<td>Sets &quot;seed&quot; of random number generator to <code>x</code>.</td>
</tr>
</tbody>
</table>
Let's consider an example `rolldice.py`
Example of Using random Library

"""
File: rolldice.py
------------------
Simulate rolling two dice
"""

```python
import random

NUM_SIDES = 6

def main():
    # setting seed is useful for debugging
    # random.seed(1)
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Dice have " + str(NUM_SIDES) + " sides each.")
    print("First die: " + str(die1))
    print("Second die: " + str(die2))
    print("Total of two dice: " + str(total))
```

"""
Today’s Goal

1. Understanding arithmetic expressions
2. Using constants
3. Random number generation
Putting it all together: dicesimulator.py
```python
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))
```
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))
    die1 10
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))
    die1 10

    die1 in main() starts as: 10
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

    die1 = 10

    print("die1 in main() starts as: 10")
def main():
    def roll_dice():
        die1 = random.randint(1, NUM_SIDES)
        die2 = random.randint(1, NUM_SIDES)
        total = die1 + die2
        print("Total of two dice: " + str(total))

    die1 in main() starts as: 10
def main():

    def roll_dice():
        die1 = random.randint(1, NUM_SIDES)
        die2 = random.randint(1, NUM_SIDES)
        total = die1 + die2
        print("Total of two dice: " + str(total))

    die1 in main() starts as: 10
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Total of two dice: " + str(total))

die1 2  die2 5  total

die1 in main() starts as: 10
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice() 
    roll_dice() 
    roll_dice() 
    print("die1 in main() is: " + str(die1))

def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Total of two dice: " + str(total))

die1 2 die2 5 total 7

die1 in main() starts as: 10
def main():
    def roll_dice():
        die1 = random.randint(1, NUM_SIDES)
        die2 = random.randint(1, NUM_SIDES)
        total = die1 + die2
        print("Total of two dice: " + str(total))

    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

die1 in main() starts as: 10
Total of two dice: 7
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

    die1 10

die1 in main() starts as: 10
Total of two dice: 7
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

    die1  10

die1 in main() starts as: 10
Total of two dice: 7
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Total of two dice: " + str(total))
def main():
    die1 = 10  # Assigning 10 to die1 in main()
    print("die1 in main() starts as: " + str(die1))
    roll_dice()  # Calling roll_dice function
    roll_dice()  # Calling roll_dice function again
    roll_dice()  # Calling roll_dice function once more
    print("die1 in main() is: " + str(die1))

def roll_dice():
    die1 = random.randint(1, NUM_SIDES)  # Randomly assigning a number between 1 and NUM_SIDES to die1
    die2 = random.randint(1, NUM_SIDES)  # Randomly assigning a number between 1 and NUM_SIDES to die2
    total = die1 + die2  # Calculating the total of two dice
    print("Total of two dice: " + str(total))

die1 in main() starts as: 10
Total of two dice: 7
```python
def main():
    die1 = 10
    print("die1 in main() starts as: "+str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: "+str(die1))

    def roll_dice():
        die1 = random.randint(1, NUM_SIDES)
        die2 = random.randint(1, NUM_SIDES)
        total = die1 + die2
        print("Total of two dice: "+str(total))

    die1 1  die2 3  total 

    die1 in main() starts as: 10
    Total of two dice: 7
```
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print("Total of two dice: " + str(total))

    die1 = 1
    die2 = 3
    total = 4

    die1 in main() starts as: 10
    Total of two dice: 7
What's Going On?

```python
def main():
    die1 = 10
    print(f"die1 in main() starts as: {die1}")
    roll_dice()
    roll_dice()
    roll_dice()
    print(f"die1 in main() is: {die1}")

def roll_dice():
    die1 = random.randint(1, NUM_SIDES)
    die2 = random.randint(1, NUM_SIDES)
    total = die1 + die2
    print(f"Total of two dice: {total}" + str(total))

die1  1  die2  3  total  4

die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
```
```python
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

    die1  10

die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
```
def main():
    die1 = 10
    print("die1 in main() starts as: "+str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: "+str(die1))

    die1 10

die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))

    die1 10

die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
Total of two dice: 5
```python
def main():
    die1 = 10
    print("die1 in main() starts as: " + str(die1))
    roll_dice()
    roll_dice()
    roll_dice()
    print("die1 in main() is: " + str(die1))
    die1
```

```
die1 in main() starts as: 10
Total of two dice: 7
Total of two dice: 4
Total of two dice: 5
die1 in main() is: 10
```
You're rockin' it!