FEBRUARY 3RD, 2025

Introduction to Functions

CE 311K - L10

CE 311K Touchpoint Survey

Built-In Function Inventory

```
print(), round(), abs(), help()
```

```
type(), isinstance(), int(), str(), float(), bool(), list(), set(), tuple(), dict()
```

```
len(), range() #, enumerate()
```

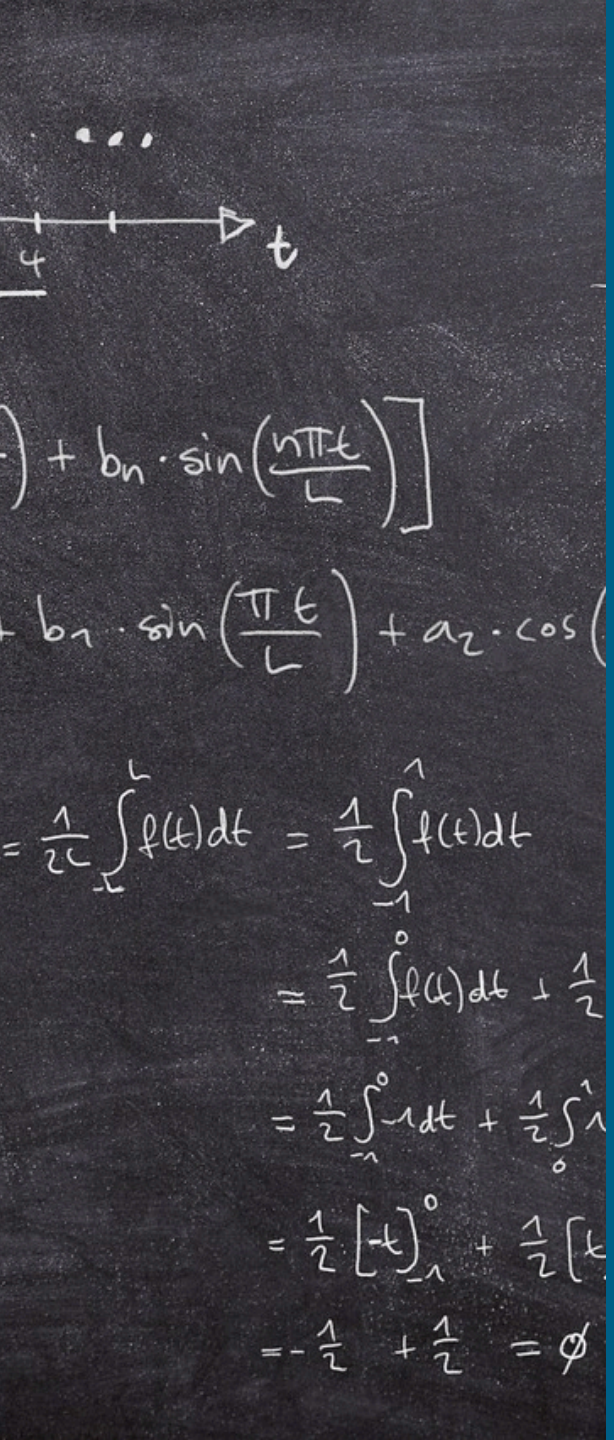
For Loop

Create a list, *temperatures,* containing these values: 200, 180, 160, 140, 120

Use the *range()* and *len()* functions to loop through each index of temperatures

In each iteration of the loop, use the index to reassign each item in temperatures to 20 less than the original value

Output the value of temperatures once the loop has completed



Functions

Functions are reusable blocks of code that perform a specific task - "mini-programs" within your program

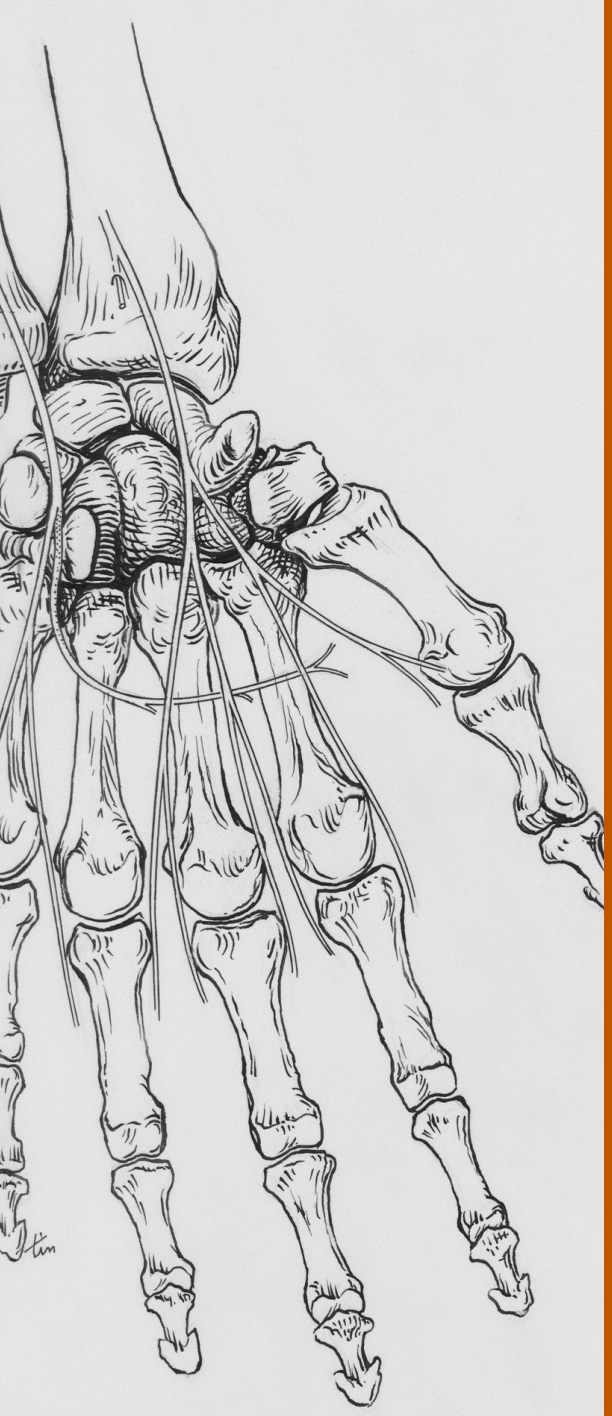
Simplify code by breaking it into smaller, logical chunks

Promote reusability and efficiency

Improve readability and maintainability

Easier testing and debugging

Generally, functions have input **parameters,** perform tasks with them, and then **return** outputs



Anatomy of a Function

```
def function_name(parameters):
    # code block
    return value
```

---

def keyword indicates a function block

Name of the function follows *def*, which adhere to standard variable naming rules

Parameters, or inputs, are in parenthesis immediately following the name - **these are optional!**

The **function body** is indented on the next line

The *return* keyword defines the output(s) from the function - **it is optional!**

Writing a Function

Create a function called get_kinetic_energy(), with one input parameter: v

Within the function, calculate the kinetic energy, ke, using 80 for the mass:

$$ke = \frac{1}{2}mv^2$$

Use a *return* statement to send back ke

Calling a Function

```
def add_ten(x):
    total = x + 10
    return total
```

Call a function by using its name and providing **arguments** if the function specifies parameters

```
result = add_ten(3)
print(result) # outputs 13
```

Parameters: specified in the function definition

Arguments: values passed to the parameters of a function

Calling a Function

Create a list of 3 velocities in m/s i.e. 10, 12, 15

Loop through each of these velocities, calling the get_kinetic_energy() function you created earlier with each velocity. Be sure to save the kinetic energy on each iteration in a temp variable.

Use a print() statement within the loop to output the kinetic energy on each iteration

Scope

Variables created inside a function are called **local variables**

These variables are only accessible within that function

Scope refers to the region of a program where a variable is accessible

Within a function (local scope) or throughout the program (global scope)

```
x = 10  # Global variable
def add_five(y):
    x = y + 5  # Local variable (different from global 'x')
    return x

print(add_value(3))  # Output: 8
print(x)  # Output: 10 (global 'x' is unchanged)
```

Once the function has finished running, its local variables are discarded

Functions: Multiple Inputs

Order of arguments matches parameters

```
def calc_area_trapezoid(b1, b2, h):
    area = 0.5 * (b1 + b2) * h
    return area

area1 = calc_area_trapezoid(2, 4, 6) # 18
area2 = calc_area_trapezoid(4, 6, 2) # 10
```

You can use **keyword arguments** to avoid confusion and ordering

```
def calc_volume(length, width, height):
    v = length * width * height
    return v

v = calc_volume(width=2, height=4, length=1) # 8
```

Creating a Function

Update your previous get_kinetic_energy() function so that now it accepts two input parameters: m and v

Update your calculation of ke so that it uses m and v, removing the default value for m we used earlier

Call your function with values: m=75 and v=13

Output the result from calling your function

Functions: Multiple Outputs

```
def quadratic_formula(a, b, c):
    discriminant = b**2 - 4*a*c

    if discriminant < 0:
        return None, None

    x1 = (-b + math.sqrt(discriminant)) / (2 * a)
    x2 = (-b - math.sqrt(discriminant)) / (2 * a)

    return x1, x2
```

Returned value can be saved to one variable (tuple) or unpacked

```
zeros = quadratic_formula(a=1, b=-3, c=2) # zeros=(2.0,1.0)
z1, z2 = quadratic_formula(a=1, b=-7, c=12) # z1=3.0, z2=4.0
```

Summary

Functions are contained mini-programs

Own state, reusable, modular, increase ease of reading/debugging

Create with *def* and a name

Parameter(s) are defined in parenthesis

Output(s) are variables/statements after *return*

Call a function by using its name with arguments for each parameter

Use keyword arguments to avoid confusion