Introduction to

PYTHON PROGRAMING

FUNCTIONS AND LOOPS

1. Functions are the building blocks of almost every Python program.
2. Functions break code into smaller chunks.
3. Functions are values and can be assigned to a variable (In Python programming).
4. You must call the function to execute it.
5. An argument is a value that gets passed to the function as input. Some functions can be called with no arguments, and some can take as many arguments as you like.
6. When a function is done executing, it returns a value as output. The return value usually — but not always — depends on the values of any arguments passed to the function.

FUNCTIONS AND LOOPS

The process for executing a function can be summarized in three steps:

1. The function is called, and any arguments are passed to the function as input.
2. The function executes, and some action is performed with the arguments.
3. The function returns, and the original function call is replaced with the return value.

FUNCTIONS CAN HAVE SIDE EFFECTS

1. You've learned how to call a function and that they return a value when they are done executing. Sometimes, though, functions do more than just return a value.
2. When a function changes or affects something external to the function itself, it is said to have a side eect.

WRITING FUNCTIONS

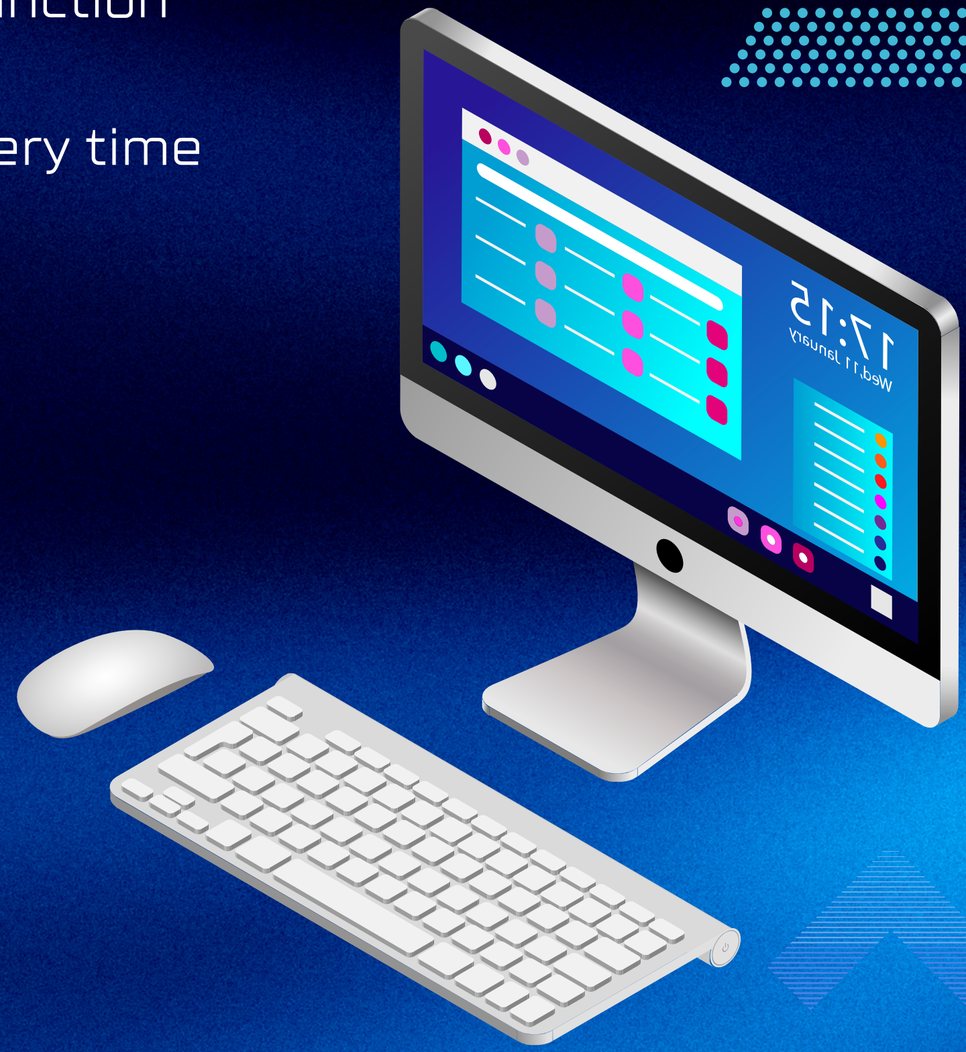
1. As you write longer and more complex programs, you may find that you need to use the same few lines of code repeatedly.
2. You need to calculate the same formula with different values several times in your code.
3. Repetitive code can be a nightmare to maintain. If you find a mistake in some code that's been copied and pasted all over the place, you'll end up having to apply the fix everywhere the code was copied. That's a lot of work, and you might miss a spot!

THE ANATOMY OF A FUNCTION

1. The function signature defines the name of the function and any inputs it expects.
2. The function body contains the code that runs every time the function is used.

```
def multiply(x, y):  # Function signature
    # Function body
    product = x * y
    return product
```

```
multiply(2, 4)
```



FUNCTIONS WITH NO RETURN STATEMENT

```
def greet(name):
    print(f"Hello, {name}!")
```

```
>>> greet("Dave")
Hello, Dave!
```

REVIEW QUESTIONS:

1. Write a function called cube() with one number parameter and returns the value of that number raised to the third power. Test the function by displaying the result of calling your cube() function on a few different numbers.
2. Write a function called greet() that takes one string parameter called name and displays the text "Hello <name>!", where <name> is replaced with the value of the name parameter.

CHALLENGE: CONVERT TEMPERATURES

1. Write a function named convert_cel_to_far() which takes one float parameter representing degrees Celsius and returns a float representing the same temperature in degrees Farenheit.
2. Write a function named convert_far_to_cel() which take one float parameter representing degrees Fahrenheit and returns a float representing the same temperature in degrees Celsius.

```
Enter a temperature in degrees F: 72
72 degrees F = 22.22 degrees C

Enter a temperature in degrees C: 37
37 degrees C = 98.60 degrees F
```

LOOPS

A loop is a block of code that gets repeated over and over again either a specified number of times or until some condition is met. There are two kinds of loops in Python: while loops and for loops.

WHILE LOOP

1. while loops repeat a section of code while some condition is true.
2. There are two parts to every while loop:
    a. The while statement starts with the while keyword, followed by a test condition, and ends with a colon (:).
    b. The loop body contains the code that gets repeated at each step of the loop.
3. When a while loop is executed, Python evaluates the test condition and determines if it is true or false. If the test condition is true, then the code in the loop body is executed. Otherwise, the code in the body is skipped and the rest of the program is executed.
4. If the test condition is true and the body of the loop is executed, then once Python reaches the end of the body, it returns to the while statement and re-evaluates the test condition. If the test condition is still true, the body is executed again. If it is false, the body is skipped.
5. This process repeats over and over until the test condition fails, causing Python to loop over the code in the body of the while loop.

```
>>> n = 1
>>> while n < 5:
...     print(n)
...     n = n + 1
...
1
2
3
4
```

HOME ABOUT MORE

Step #	Value of n	Test Condition	What Happens
1	1	1 < 5 (true)	1 printed; n incremented to 2
2	2	2 < 5 (true)	2 printed; n incremented to 3
3	3	3 < 5 (true)	3 printed; n incremented to 4
4	4	4 < 5 (true)	4 printed; n incremented to 5
5	5	5 < 5 (false)	Nothing printed; loop ends.

THE INFINITE LOOP

If you aren't careful, you can create an innite loop. This happens when the test condition is always true. An infinite loop never terminates. The loop body keeps repeating forever.

```
>>> n = 1
>>> while n < 5:
...     print(n)
...
```

FOR LOOP

- A for loop executes a section of code once for each item in a collection of items. The number of times that the code is executed is determined by the number of items in the collection.
- The for loop has two main parts:
  - 1. The for statement begins with the for keyword, followed by a membership expression, and ends in a colon (:).
  - 2. The loop body contains the code to be executed at each step of the loop, and is indented four spaces.

FOR LOOP


```
for letter in "Python":
    print(letter)
```

- In this example, the for statement is for letter in "Python". The mem-bership expression is letter in "Python".
- At each step of the loop, the variable letter is assigned the next letter in the string "Python", and then the value of letter is printed.
- The loops runs once for each character in the string "Python", so the loop body executes six times.

NESTED LOOPS

- As long as you indent the code correctly, you can even put loops inside of other loops.
- A loop inside of another loop is called a nested loop, and they come up more often than you might expect.
- Nesting loops inherently increases the complexity of your code, as you can see by the dramatic increase in the number of steps run in the previous example compared to examples with a single for loop.
- Using nested loops is sometimes the only way to get something done, but too many nested loops can have a negative effect on a program's performance.

```
for n in range(1, 4):
    for j in range(4, 7):
        print(f"n = {n} and j = {j}")
```

REVIEW QUESTIONS:

1. Write a for loop that prints out the integers 2 through 10, each on a new line, by using the range() function.
2. Use a while loop that prints out the integers 2 through 10 (Hint: You'll need to create a new integer first.)
3. Write a function called doubles() that takes one number as its input and doubles that number. Then use the doubles() function in a loop to double the number 2 three times, displaying each result on a separate line. Here is some sample output:

```
4
8
16
```

THANK YOU