

Unit 1

Unit 2

Unit 3

Unit 4

Unit 5

Unit 6

Unit 7

Unit 8

Unit 9

Unit 10

Name: Harbans Singh

Class: Programming & Prototyping






Guidebook





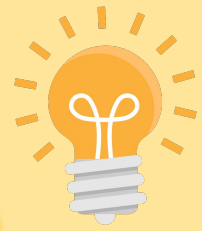
1. All python errors and troubleshooting
2. ProjectStem
3. Figma
4. PythonTutor
5. Flow Chart (Make copy)

Flow chart symbols

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision



to cover



Unit 1

Unit 1: Intro to Python

1.1 What is Computer Science?

1.2 Using Python

1.3 First Program

1.4 Hardware and Software

1.5 Output

1.6 Input

1.7 Data Types and Variables

1.8 Analog vs. Digital

1.9 Understanding Binary

Unit 1 Vocabulary

Assignment 1: Silly Sentences / Test 1

Unit 1 Review



1. **Computer Science** - The study of principles and use of computers
2. **Computer** - An electronic device consisting of hardware and software
3. **CPU** - Central Processing Unit: carries out program instructions
4. **Hardware** - The physical machine, anything you can touch
5. **Software** - Programs that run on hardware
6. **Coding** - Using a formal language in order to create software
7. **Main Memory** - Short term memory, deletes after shut off
8. **Secondary Memory** - Long-term memory, storage
9. **Program** - Instructions that a computer follows, written in code
10. **Input** - The user sends information to the computer
11. **Output** - The computer sends information to the user
12. **Data Type** - All values in a programming language have a "type" - such as Number, Boolean, or String - that dictates how the computer will interpret it, For example $7+5$ is interpreted differently from $"7"+"5"$.
13. **Expression** - Any valid unit of code that resolves to a value.
14. **Variable** - A placeholder for a piece of information that can change
15. **Compiler** - A program that converts commands so that a computer can understand and execute them
16. **Integrated Development Environment (IDE)** - Software or an application that combines multiple tools in one window
17. **Print** - The Python command that displays text and numbers on the screen
18. **Syntax** - The rules that define the written structure of a programming language



Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs. Javascript is more advanced and used for big software projects.

Variables:

They need to be declared and contains memory. There are 3 types of variables

Boolean: T/F

Number: Values, no "", can use mathematical operators - Int: whole # - Float: Decimal #

String: Cannot be evaluated, must be enclosed in ""


- Variable names should indicate their purpose
- Can contain letters, numbers, and underscores
 - Use underscores to separate words
 - CamelCase
- Python is CASE SENSITIVE (e.g. Numkids vs. numkids)
- No spaces symbols or keywords:
 - Keywords like: print, false, true, else, for, while
- Can't start with a Number (e.g. 1Game vs Game1)
- Not be too long

To provide useful output, text is combined with variables using concatenations

This is a typeerror since you Went from a string to an Integer which can't be printed

Reassigning Variable Values

```
x = "hello"  
x = 5  
print("Oh, "+x+"!")
```



TYPEERROR!!

Cannot concatenate 'str and 'int' objects



Type	Description	Examples
string	Texts, words	"Hello world123"
int	Whole numbers and their opposites	3, -8, 90, 0 -10000
Float	+/- numbers with decimals	1.34, 6.0
boolean	Only 2 values	True False

NameError: name 'c' is not defined

```
1 a="hello"
2 b=5
3 c
```

SyntaxError: EOL while scanning the string literal

```
a = "hello"
b = 5
c = "5"
```

Type Coercion

string	str()	str(5) = "5" str("hey") = "hey"
integer	int()	int("4") = 4 int("hey") = ERROR

1. List the three types of software

Operating system, compiler, application

2. What is the CPU?

The CPU is the central processing unit for a computer. Brains of the computer, carries out programs instructions

3. List the 5 things all computing devices have in common.

1. CPU (central processing unit)
2. Main Memory (storage device)
3. Secondary memory (storage device)
4. Input device (camera, mouse, keyboard)
5. Output device (screen, speaker, audio)

4. What are programs?

Programs are the instructions made out of a formal language that uses code for the computer to follow

5. What is the difference between hardware and software?

Hardware is the physical components of a computer and software is the inside of a computer (programs)

Unit 1 CS Python Fundamentals

6. What is output by the following?

a. `print("Computer \n\t\tScience")`

Computer

Science

b. `print("Blue ") print("Sky")`

Blue

Sky

7. Write the code to print the following, be sure to include the quotes. Use only one print command.

"Success consists of going from failure to failure without loss of enthusiasm."

- Winston Churchill

`print(" \"Success consists of going from failure to failure without loss of enthusiasm.\"\\n\t-Winston Churchill")`

8. What symbol do we use for comments?

#

9. Define Variable.

A place in the computer memory that you name so you can save and retrieve data

10. What are the two main data types in Python?

- A) Number
 - a) int() they are whole numbers
 - b) floats() they are decimal numbers
- B) String ' or ''
- C) Boolean True or False

11. What is the difference between analog and digital?

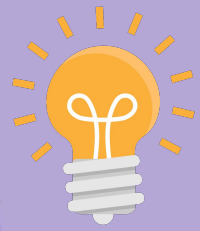
Analog is data in the real world, while digital is information converted into numerical data to be stored and manipulated by a computer

12. Write the code to input a number.

```
age = int(input("how old are you in numbers?"))
```

13. What does the `str` command do?

Changes the number into a string so it can be combined with other strings



Unit 2: Number Calculations & Data

2.1 Computer History

2.2 Basic Calculations

2.3 Modular Division

2.4 Built-in Functions

2.5 Random Numbers

2.6 Big Data

2.7 Working With a Real Data Set

Unit 2 Vocabulary

Assignment 2: Room Area

Test 2 Quiz 2

Unit 2 Review

VOCABULARY:

- To create multi-line comments you can use “...” over and under the comment you wrote

Function	Description	Example	Output
<code>sqrt(x)</code>	Takes the square root of x and returns a float	<pre>import math print(math.sqrt(81))</pre>	9.0
<code>fabs(x)</code>	Takes the absolute value of x and returns it as a float	<pre>import math print(math.fabs(-9))</pre>	9.0
<code>pow(x, y)</code>	Raises x to the y power and returns it as a float	<pre>import math print(math.pow(3, 2))</pre>	9.0

```
import random
```

Here are some useful functions in the random module:

Function	Sample Usage	Description
<code>randint(x, y)</code>	<pre>import random x = random.randint(5, 10)</pre>	Returns a random integer between x and y, inclusive.
<code>random()</code>	<pre>import random x = random.random()</pre>	Returns a random number from [0.0, 1.0) (i.e., greater than or equal to 0.0 but less than 1.0).
<code>choice(sequence)</code>	<pre>import random animal = random.choice(["cat", "dog", "fish", "snake"])</pre>	Picks a random element of a sequence. (In this example, animal could be randomly assigned to be cat, dog, fish, or snake). The sequence must always be contained within brackets.

NOTES:

- To create multi-line comments you can use “...” over and under the comment you wrote

`max()` returns the biggest element in a set of data and `min()` returns the minimum.

For example:

```
print(min("cat", "dog", "iguana", "anteater", "fish", "aardvark"))
```

will print out the minimum based on alphabetical order (so it would print out aardvark).

```
import simpleplot

dataset1 = [(1, 4), (1, 5), (2, 7), (4, 9)]
dataset2 = [(1, 2), (2, 7), (2, 5), (7, 6)]

simpleplot.plot_lines("Sample", 400, 300, "x", "y", [dataset1, dataset2], True, ["Dataset 1", "Dataset 2"])
```

- `max()`
- `min()`
- `simpleplot`
- `plot_lines()`



Openers & Ideas

[to divider](#)

Python provides the capability to perform basic arithmetic operations such as addition and multiplication

Parenthesis	(....)	Exponents	**
Multiplication	*	Division	/
Addition	+	Subtraction	-

For more advanced operations, you must import the **math** library using

```
1 import math
```

Square Root	math.sqrt(x)	Trig	math.sin(x)
Abs Value	math.fabs(x)	Degrees	math.degrees(x)
Log	math.log(x,base)	Pi	math.pi

Code

```

1 import math
2 r = int(input("What is the radius of your circle?"))
3 C = 2*r*math.pi
4 A = math.pi*r**2
5 print("C=", C, " A =", A)

```

Output

C= 31.41592653589793 A = 78.53981633974483

Code

```

1
2
3 length = int(input("what is the length of the rectangle?"))
4 width = int(input("what is the width of the rectangle?"))
5 perimeter = 2 * (length + width)
6
7 print("The perimeter is " + str(perimeter))
8

```

Output

The perimeter is 12

Unit 2 Test Review

1. List the four basic number operations in programming.
Multiplication (*), division (/), addition (+), subtraction (-)
2. What data type do we use to store numbers that will be used for calculations?
Integers
3. Why do we need to know the data types in programming?
So the computer knows what to do with the data
4. Why do we use number calculations in programming?
It can make our human life easier, and it is faster than a human
5. What does the % do?
It divides 2 numbers and outputs the remainder
6. What is:
 - a. $4 \% 3 = 1$
 - b. $110 \% 2 = 0$
 - c. $43 \% 2 = 1$
 - d. $3 \% 12 = 3$
 - e. $15 \% 10 = 5$
7. What is the Python math function for each of the following?
 - a. Exponent = `math.pow()`
 - b. Square Roots = `math.sqrt()` or `**0.5`
 - c. Absolute Value = `abs()` or `fabs()`
8. Define function.
A line of code that is run when told to/ collection of commands
9. Write a program to input two numbers and find the quotient (divide).

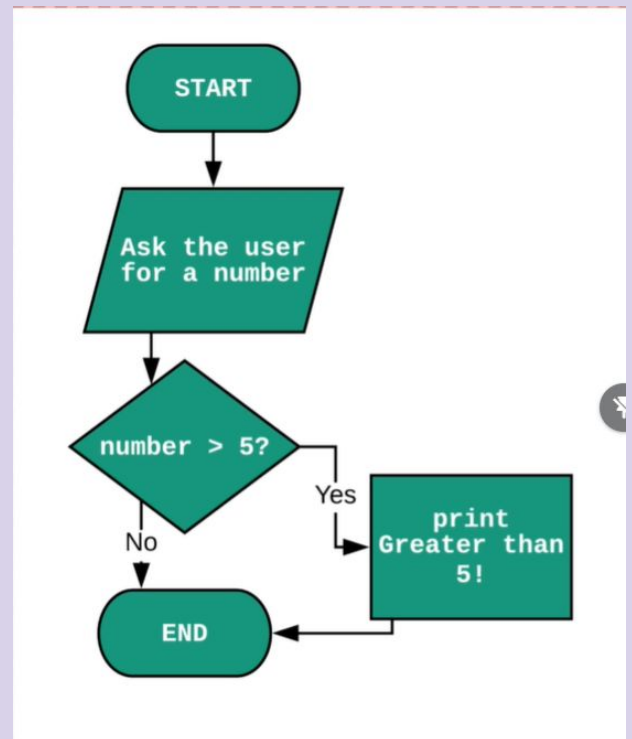
```
dividend = int(input("What is the dividend?"))  
divisor = int(input("What is the divisor?"))  
print(dividend/divisor)
```

What happens if you input 0 for the second number?
The quotient is zero

10. What is output: `print (7 * 3 - 4 + 3**2)`
26

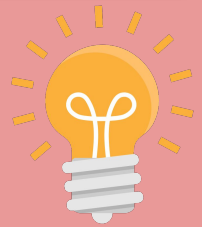
NOTES:

- To create multi-line comments you can use “...” over and under the comment you wrote





[to cover](#)



Unit 3

Unit 3: Making Decisions

3.1 Functions

3.2 Simple Ifs

3.3 Logical Operators

3.4 If – Else

3.5 Else – If

3.6 Defining Algorithms

3.7 Algorithm Challenge

Vocabulary Practice.

Assignment 3: Chatbot

Test 3 Quiz 3

[Unit 3 Review](#)

Comparison Operators

Operators	Description	Example
==	Equal	5 == 5 (true)
!=	Not Equal	4 != (true)
> >=	Greater than Greater than or equal to	3 > 1 (true) 3 >= 3 (true)
< <=	Less than Less than or equal to	1 < 3 (true) 3 >= 3 (true)

Operators	Description	Examples
and	True only when all conditions are true	5 > 3 and 4 < 5 (true) 2 > 1 and 3 < 1 (false)
or	True when at least one condition is true	4 == 4 or 3 == 2 (true)
not	True when false	Not 3 > 2 (False) Not 28 < 9 (true)

Conditionals Explained

Conditional	Description
if:	Starts the conditional with first comparison
elif:	Starts the next comparison if it doesn't count on if
else:	Happens if it doesn't count with others

NOTES: CONDITIONALS



to divider

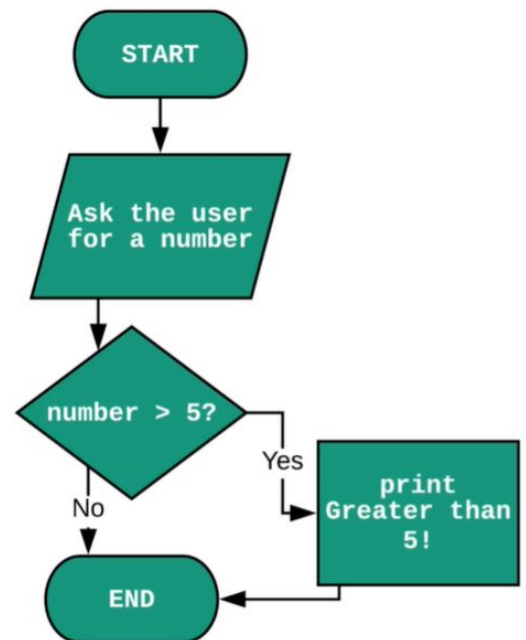
NOTES:

```
num = int(input("what is ur number"))
```

```
if num > 5:
```

```
    print("Greater than 5")
```

If number is greater than 5 then
it will send the message



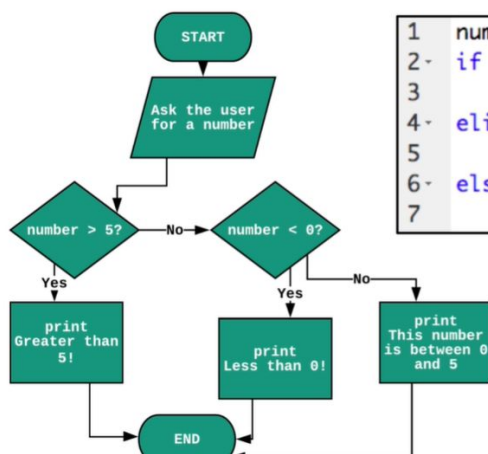
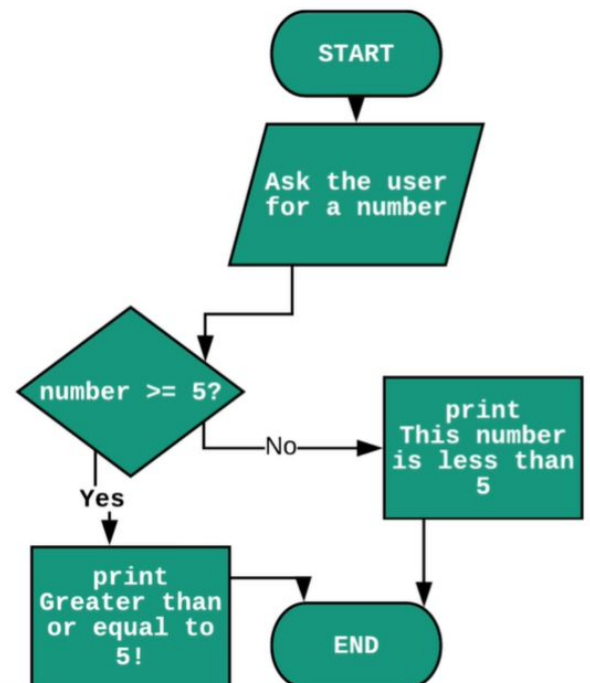
```
num = int(input("give me a number"))
```

```
if num >= 5:
```

```
    print("this number is greater  
    than or equal to 5")
```

```
else:
```

```
    print("This number is less than  
    5!")
```



```
1 num = int(input("Give me a number: "))
2 if num > 5:
3     print("Greater than 5!")
4 elif num < 0:
5     print("Less than 0!")
6 else:
7     print("This number is between 0 and 5")
```

Function Syntax

Defines the function

```
Def mult_function(x,y):  
    Z = x * y  
    print("I heart math")  
    return z
```

Need to use colon

**Returns the value to
what it was before**

```
mult_function(5,4)
```

**Calls function with the 2 numbers as
the variables**

Functions are something that is very useful to use in long code. Functions can be written out once and then be called in the future just by the name you gave to it.

This can help you to save time and space when coding long tedious applications.

Unit 3 Review



[to divider](#)

global line allows you to use a variable inside and outside of a function

for i in range(x) = uses the value of x for how many times the stuff below should occur

return allows you to use a function and allows you to send back specific information to be used outside of the function in a variable

```
for i in range(rolls): #allows you to use the number
    def random_roll():
        global score #allows you to use the variable in a function
        randNum = random.randint(1,6)
        userNum = int(input("Choose a number 1-6"))
        if randNum == userNum:
            score = score + 6
            print("Correct! +6 points")
        else:
            score = score - 1
            print("Wrong! -1 point")
    random_roll()
```

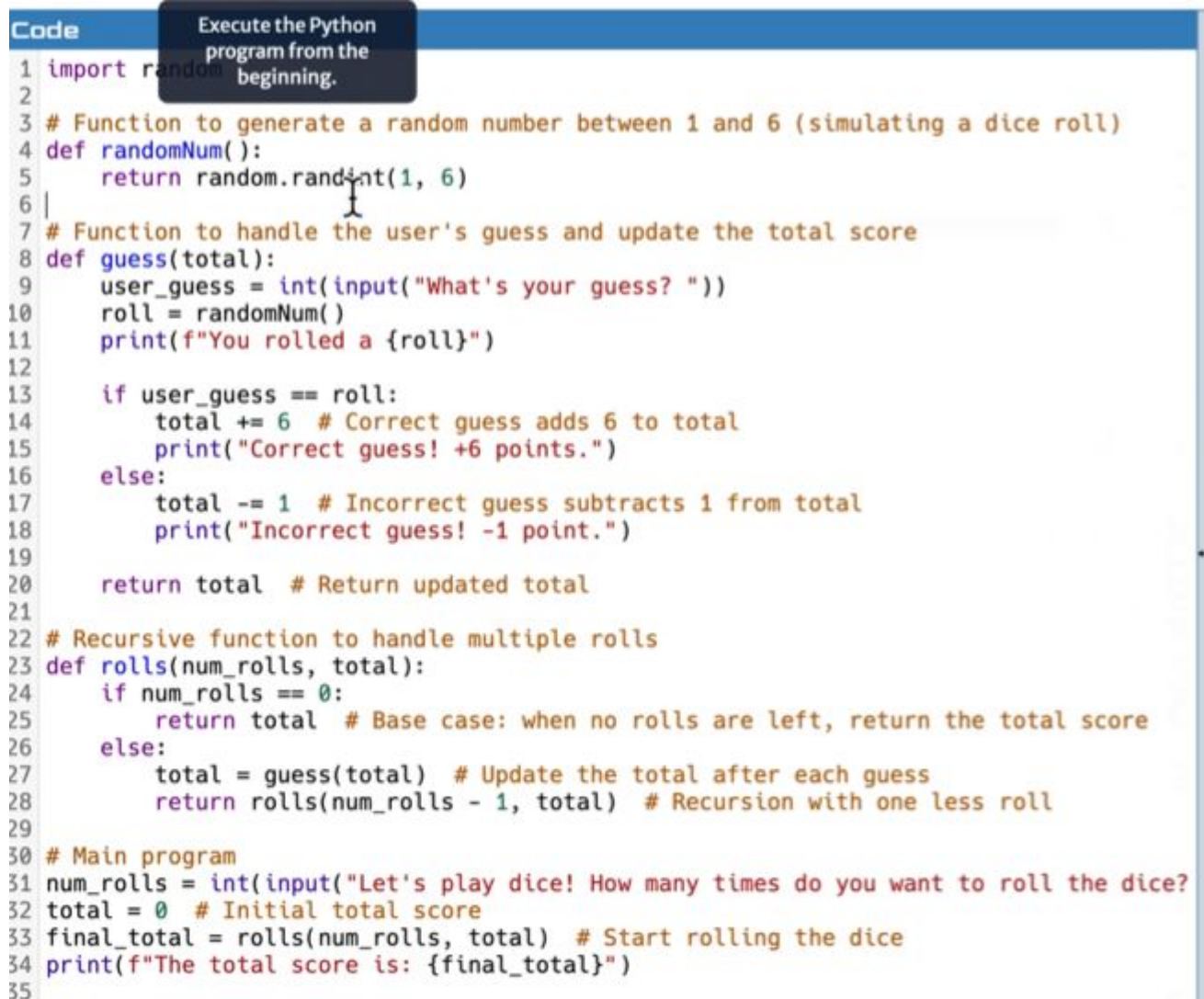
If **rolls** is 4 the code will run 4 times

An algorithm is a set of instructions. It can be in code and analog like a recipe to make food

They can have different kind of algorithms as there are a lot things that can be different like the time it takes to do, how efficient and good it is.

Import random

```
def randomNum():  
    return random.randint(1,6)  
def guess(total):  
    user_guess = int(input("What's your guess? "))  
    roll = random()  
    print(f"You rolled a {roll}")
```



```
Code  
Execute the Python program from the beginning.  
1 import random  
2  
3 # Function to generate a random number between 1 and 6 (simulating a dice roll)  
4 def randomNum():  
5     return random.randint(1, 6)  
6  
7 # Function to handle the user's guess and update the total score  
8 def guess(total):  
9     user_guess = int(input("What's your guess? "))  
10    roll = randomNum()  
11    print(f"You rolled a {roll}")  
12  
13    if user_guess == roll:  
14        total += 6 # Correct guess adds 6 to total  
15        print("Correct guess! +6 points.")  
16    else:  
17        total -= 1 # Incorrect guess subtracts 1 from total  
18        print("Incorrect guess! -1 point.")  
19  
20    return total # Return updated total  
21  
22 # Recursive function to handle multiple rolls  
23 def rolls(num_rolls, total):  
24     if num_rolls == 0:  
25         return total # Base case: when no rolls are left, return the total score  
26     else:  
27         total = guess(total) # Update the total after each guess  
28         return rolls(num_rolls - 1, total) # Recursion with one less roll  
29  
30 # Main program  
31 num_rolls = int(input("Let's play dice! How many times do you want to roll the dice?"))  
32 total = 0 # Initial total score  
33 final_total = rolls(num_rolls, total) # Start rolling the dice  
34 print(f"The total score is: {final_total}")  
35
```

The conditions in the *if* statements we've been using so far are examples of **boolean expressions**: expressions that evaluate to either True or False. However, we have the ability to check for multiple conditions if we use a **logical operator**.

For example:

```
n = int(input("Enter a number: "))

if n >= 0 and n <= 100:
    print("Grade is valid")
    print("No error detected\n")

print("Done")
```

In this example, we used the logical operator **and** in order to check for multiple conditions at once.

If we want to print code when where only one of two conditions needs to be true, we can use the **or** logical operator:

```
x = -5
y = 10
if x < 0 or y < 0:
    print("At least x or y is negative")
```

If *x* is less than 0, *y* is less than 0, **or** both *x* and *y* are less than 0, then the print command will run.

The **not** logical operator takes the opposite of the boolean expression.

```
if not condition:
    print("Condition is false")
```

If *condition* were True, (*not condition*) is False, so it would not print. If *condition* were False, (*not condition*) is True so it would print. If *condition* were the expression $5 > 7$, *condition* would be False because 5 is not greater than 7. *Not condition* is the opposite, so (*not condition*) is True and the statement would print.

So far, our *if* statements let us run code when the condition evaluates to True. The *if-else* statement allows us to expand on this and specify what happens when the condition is False.

For example:

```
if condition:
    print("Condition is true")
else:
    print("Condition is false")
print("This is outside of the if-else-statement. Will print either way")
```

(If condition is True, print that it's true. **Otherwise/else**, print that it's false).

This is useful because it guarantees that something will be done. If the True block of your *if* statement isn't run, the False block will definitely run. In addition, it guarantees that only one block of code is run. These two guarantees help us cut down on coding mistakes that we might have made if we had simply used a bunch of *if* statements on their own to emulate this behavior.

We've seen the usefulness of the else part of an if statement. In fact, the if statement has yet another construct that can make our code simpler, especially if we have more than two conditions to check.

Take this code for example:

```
score = int(input("Enter your score: "))

if score >= 1000:
    print("You got three stars.")

if score >= 750 and score < 1000:
    print("You got two stars.")

if score >= 500 and score < 750:
    print("You got one star.")

if score < 500:
    print("You got zero stars.")
```

In this case, we would need four separate if statements and the conditions inside of them need to use logical operators.

The code below successfully uses else-if logic to make the code simpler:

```
score = int(input("Enter your score: "))

if score >= 1000:
    print("You got three stars.")
elif score >= 750:
    print("You got two stars.")
elif score >= 500:
    print("You got one star.")
else:
    print("You got zero stars.")
```

It is guaranteed only one of these print statements will print, because all of the conditions are chained together. It will first start at the top and stop running as soon as one of the conditions is true. Because of this, we no longer need logical operators. The code checks to see if score is greater than or equal to 1000. If that is false, we already know that it has to be less than 1000, so it is redundant to have to check that below.

1. What is an if-statement used for?
To allow programs to make decisions
2. When do you use an else?
When the if statement is not correct and need something else to happen
3. What does != mean?
Not equal
4. Test if num is between 78 and 45 inclusive.
if num <= 78 and num >= 45:
 print(str(num), "is between 78 and 45")
5. Test if x is NOT between 67 and 32.
if x >= 67 or x <= 32:
6. Test if value is positive.
if x > 0:
7. Input two numbers and print the smaller to the screen.
a = int(input("Choose a whole number"))
b = int(input("Choose a whole number"))
if a > b:
 print(b)
else:
 print(a)

Correct the mistakes (there may be more than one):

8. IF $(x < y < 8)$:

if $x < y$ and $y < 8$:

9. IF (word = "pumpernickel "):

if word == "pumpernickel":

Answer the following:

10. Who is George Boole?

George Boole is the person who created the Boolean statements such as greater than, less than, or equal to. These are important in computer science

11. Precise set of rules for how to solve a problem is called a algorithm.

12. What are five things algorithms must have?

- Have an order
- Clear instructions
- Stop in a finite amount of time
- Produce a result
- Operations that can be performed by a computer more efficiently than a human being

13. Why do we analyze algorithms?

We analyze algorithms to predict the performance and help of an algorithm

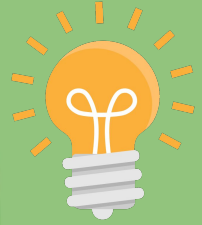
Unit 3 ExtraPage



to divider



to cover



Unit 4

Unit 4: Repetition and Loops

4.1 Loops

4.2 Count Variables

4.3 Two Ways to End a Loop

4.4 Data Revisited

4.5 Review - Looping

4.6 Range Function

4.7 For Loops

4.8 Counting by Other Than

4.9 Summing

4.10 Review of Algorithms and Tracing

4.11 Modeling and Simulation

Assignment 4: Divisible by Three

Test 4

Quiz 4

Loops:

Loops can be used with the “while” line. This code will not stop since the if statement is valid.

Ex

```
name = input("Enter a name, STOP to end")

while name != "STOP": When variable name is not STOP, it will do the code
    print("You entered: " + name)
    name = input("Enter a name, STOP to end")
print("Done.")
```

Ex 2

```
city = input("Please enter a city name: (Nope to end) ")

while city != "Nope":
    print(f"Oh! {city} is a cool spot.")
    city = input("Please enter a city name: (Nope to end) ")
```

You can use a counter variable to count how many times the loop happens

```
name = input("STOP to end: ")

c = 0

while name != "STOP":

    print("You entered: " + name)

    name = input("STOP to end: ")

    c = c + 1

print("You entered " + str(c) + " words.")
```

Ex

```
pet = 0
pets = 0

while pet != "stop":
    pet = input("What pet do you have?")
    pets = pets + 1
    if pet != "stop":
        print("You have one", pet + ".", "Total # of Pets:", pets)
```

The range function returns a set of numbers based on the information, or **parameters**, provided. For example, the parameter in `range(8)` is 8.

- `print(8)` would print a number: 8
- `print(list(range(8)))` would print a list: `[0, 1, 2, 3, 4, 5, 6, 7]`

In the example above, notice that printing `range(8)` returns a list of numbers, rather than a single number. Also notice that the list of numbers goes from 0 to 7, or '0' to 'parameter-1'.

A range function can take either 1, 2, or 3 parameters:

1. `range(x)` returns numbers from 0 to x-1. For example, `range(4)` returns a list of numbers from 0 to 3: `[0, 1, 2, 3]`
2. `range(x, y)` returns numbers from x to y-1. For example, `range(2, 8)` returns a list of numbers from 2 to 7: `[2, 3, 4, 5, 6, 7]`
3. `range(x, y, z)` returns numbers from x to y-1 counting by z. For example, `range(9, 0, -2)` returns a list of every other number between 9 to 1, including both 9 and 1: `[9, 7, 5, 3, 1]`

FOR LOOPS:

```
for x in range(5):  
    print(f"Loop #: {x}")
```

The variable `x` is local so it will only exist inside of the loop

The loop will run 5 times and will print `x` starting from 0

Ex:

```
x = 10
```

```
for i in range(4):
```

```
    x = x + 1
```

```
    print(x)
```

This will print 11, 12, 13, 14

Code	Output
1 x = 10	Loop # 0, x = 10
2 for i in range(4):	Loop # 1, x = 11
3 print(f"Loop # {i}, x = {x}")	Loop # 2, x = 12
4 x = x + 1	Loop # 3, x = 13
5	

This setup is useful for programs where you have a value which keeps changing

FOR Loops Ranges

Ranges are written in 3 different ways:

- **range(end)** (5) → 5
- **range(start, end)** → (1, 4) →
- **range(start, end, step)**

included

excluded

Where the end is
ALWAYS exclusive

Where the start is
ALWAYS inclusive

Where the step is the
amount to change by

****Inputs to ranges must be integers only****

Code

```
1  
2 for i in range(0, 101, 5):  
3     print(f"i # {i} ")  
4
```

Output

```
i # 0  
i # 5  
i # 10  
i # 15  
i # 20  
i # 25  
i # 30  
i # 35  
i # 40  
i # 45  
i # 50  
i # 55  
i # 60  
i # 65  
i # 70  
i # 75  
i # 80  
i # 85  
i # 90  
i # 95  
i # 100
```

Going up by 5 starting from 0 to 100

FOR Loops run a certain number of times defined in `range()`

```
1 for i in range(5):
2     print("hi")
```

A **for** loop can be used to sum a list of numbers. For example, this program:

```
sum = 0
for i in range(2, 6):
    sum = sum + i
    print(str(sum))
```

adds $2 + 3 + 4 + 5$ and prints:

14

WHILE Loops run an infinite number of times if their condition is true

```
1 i = 0
2 while i < 5:
3     print(i)
4     i = i + 1
```

While Loop	Outcome
<pre>1 n = 5 2 3 while n > 0: 4 print("Rinse") 5 print("Lather") 6 print("Dry off")</pre>	<p>Stopped after running 1000 steps.</p> <p>Congratulations, you've crashed your browser!</p>

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Python 3.6
[known limitations](#)

```

1 count = 0
2 while count < 5:
3     print("hi")
4     count += 0.1
5 print("Goodbye")

```

[Edit this code](#)

→ line that just executed
→ next line to execute

[<< First](#)
[< Prev](#)
[Next >](#)
[Last >>](#)

Done running (156 steps)

Print output (drag lower right corner to resize)

```

hi
hi
hi
hi
hi
Goodbye

```

Frames Objects

Global frame

```

count 5.1

```

UNIT 4 EXTRA PAGE



to divider