Name: Harbans Singh_____

Class: Programming&Prototyping

Guidebook

Unit 1

Unit 2

Unit 3

Unit 4

Unit 5

Unit 6

Unit 7

Unit 8

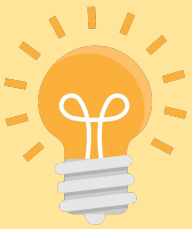
Unit 9

Unit 10

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



Unit 1:
Intro to Python

Vocabulary

to Divider

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

Information/Content

to Divider

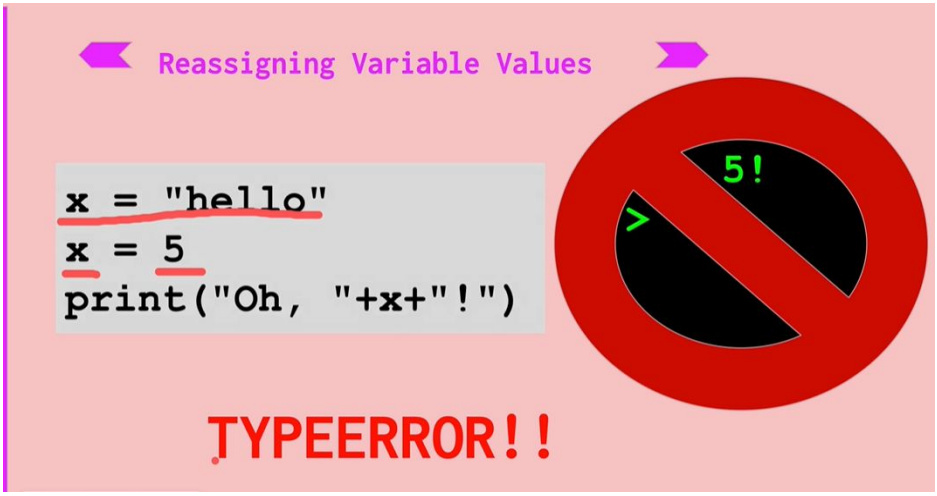
Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs. Javascript is more advanced and used for big software projects.

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



Cannot concatenate ‘str and ‘int’ objects

Information/Content

BACK to Divider

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

UNit 1 Review Questions

to divider

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

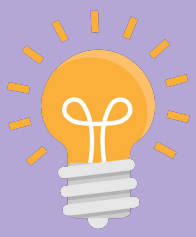
12. Write the code to input a number.

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

to cover



Unit 2: Number Calculations & Data

VOCABULARY:

- To create multi-line comments you can use "…" over and under the comment you wrote

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

```
import random
```

Here are some useful functions in the random module:

Function	Sample Usage	Description
randint(x, y)	`import random` `x = random.randint(5, 10)`	Returns a random integer between x and y, inclusive.
random()	`import random` `x = random.random()`	Returns a random number from [0.0, 1.0) (i.e., greater than or equal to 0.0 but less than 1.0).
choice(sequence)	`import random` `animal = random.choice(["cat", "dog", "fish", "snak e"])`	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()

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

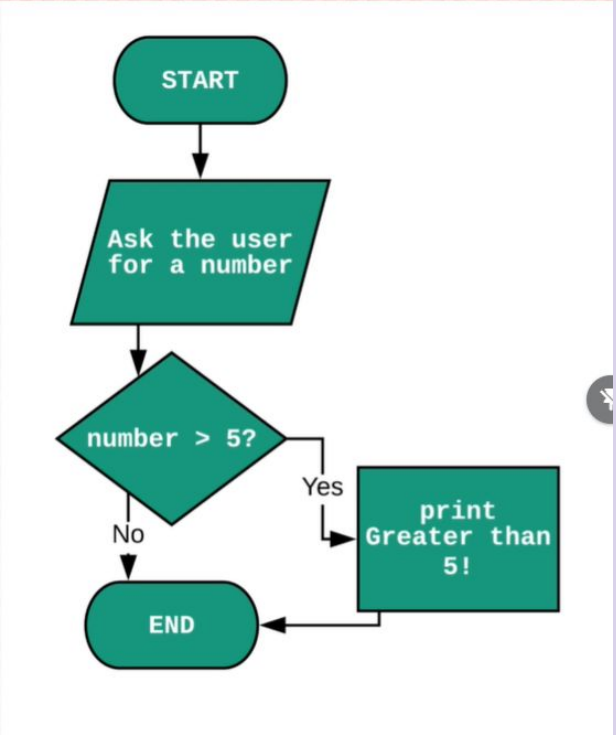
9. Write a program to input two numbers and find the quotient (divide).
```
dividend = int(input("What is the dividend?"))
divisor = int(input("What is the divisor?"))
print(dividend/divisor)
```

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

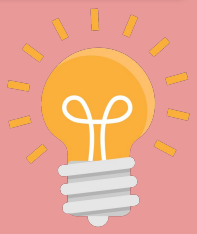
10. What is output: print (7 * 3 - 4 +3**2)
26

NOTES:

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



BACK to cover



Unit 3: Making Decisions

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

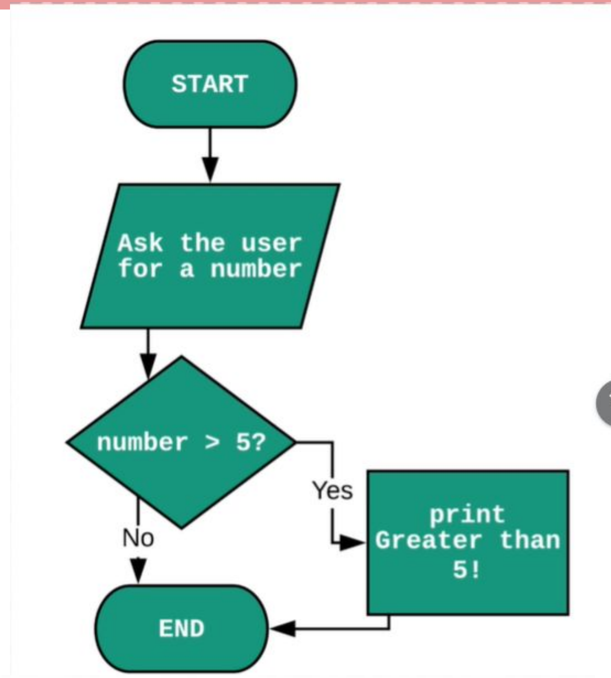
Conditional	Description
if:	Starts the conditional with first comparison
elif:	Starts the next comparison if it doesn't count on if
else:	Happens if it doesn't count with others

NOTES: CONDITIONALS

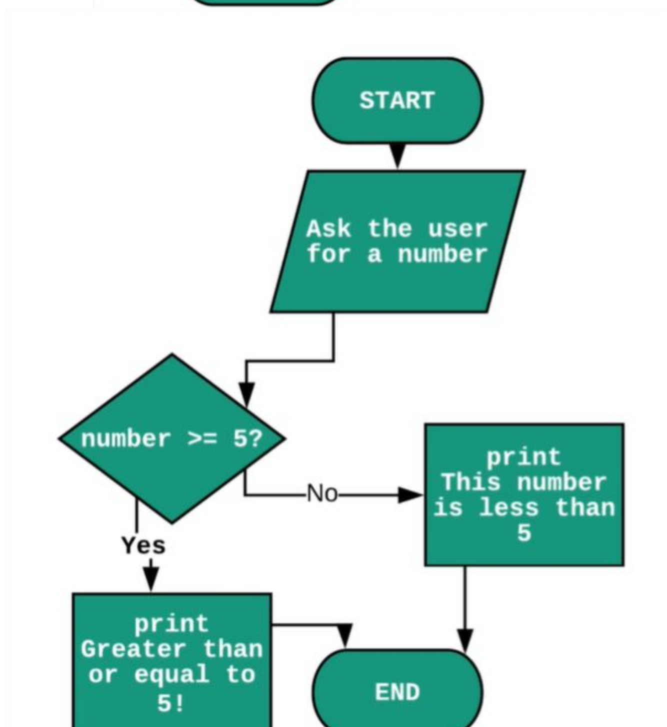
🔙 to divider

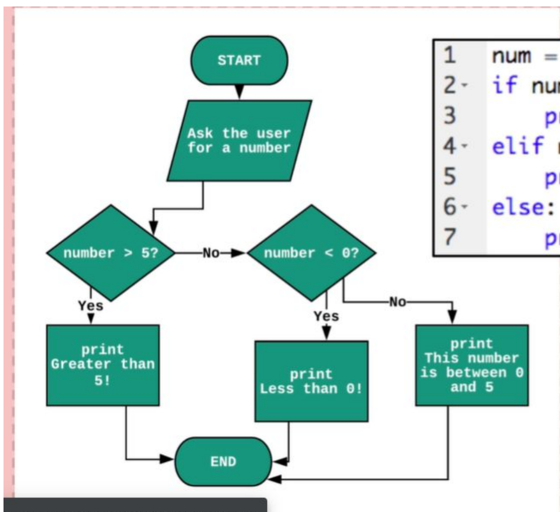
NOTES:

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!")






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

FUNCTIONS 10/17/24

to divider

Function Syntax

Defines the function

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

mult_function(5,4)
```

Need to use colon

Returns the value to what it was before

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.

NOTES: 10/18/24

BACK to divider

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.

```
import random

# Function to generate a random number between 1 and 6 (simulating a dice roll)
def randomNum():
    return random.randint(1, 6)

# Function to handle the user's guess and update the total score
def guess(total):
    user_guess = int(input("What's your guess? "))
    roll = randomNum()
    print(f"You rolled a {roll}")

    if user_guess == roll:
        total += 6  # Correct guess adds 6 to total
        print("Correct guess! +6 points.")
    else:
        total -= 1  # Incorrect guess subtracts 1 from total
        print("Incorrect guess! -1 point.")

    return total  # Return updated total

# Recursive function to handle multiple rolls
def rolls(num_rolls, total):
    if num_rolls == 0:
        return total  # Base case: when no rolls are left, return the total score
    else:
        total = guess(total)  # Update the total after each guess
        return rolls(num_rolls - 1, total)  # Recursion with one less roll

# Main program
num_rolls = int(input("Let's play dice! How many times do you want to roll the dice?
total = 0  # Initial total score
final_total = rolls(num_rolls, total)  # Start rolling the dice
print(f"The total score is: {final_total}")
```

NOTES

BACK to divider

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.

NOTES

BACK to divider

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.

Unit 3 Review

to divider

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)
```

Unit 3 Review

to divider

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

BACK to divider

to cover

Unit 4:
Repetition and Loops

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]`

10/25/24 LOOPS

BACK to divider

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

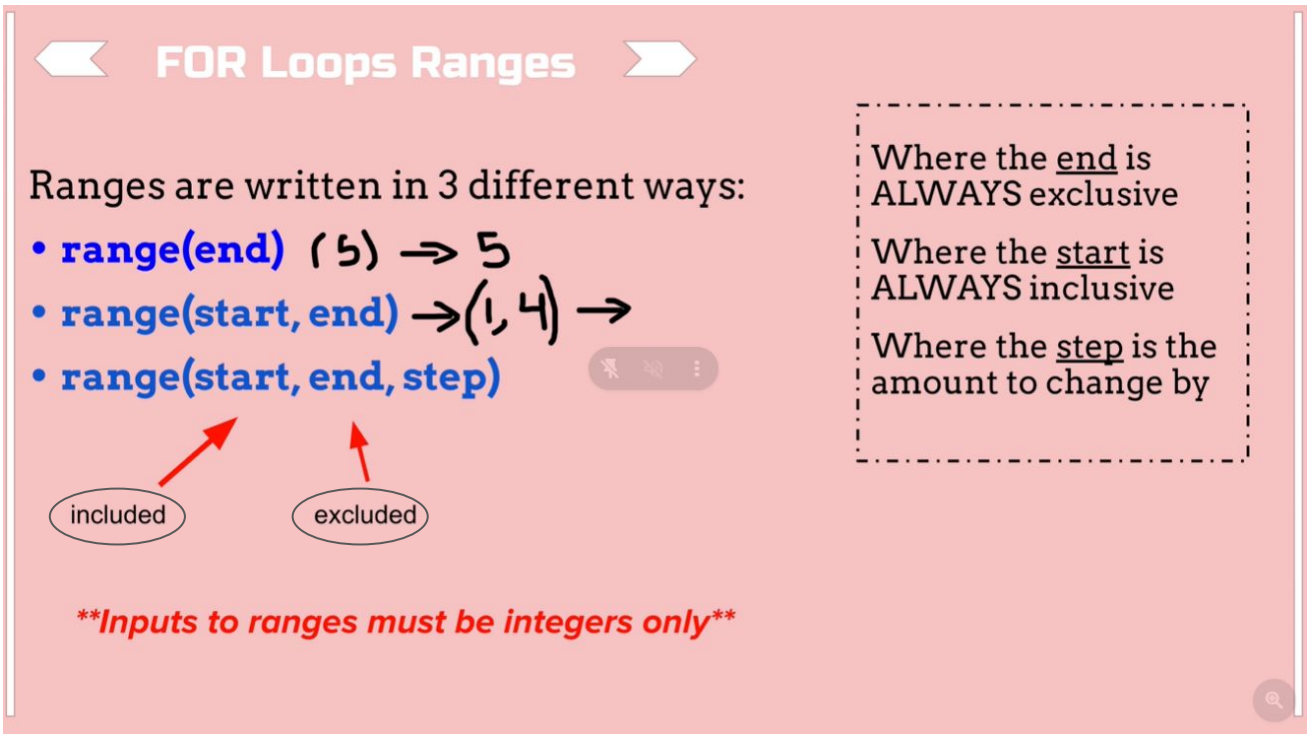
The variable x is local so it will only exist inside of the loop
The loop will run 5 times and will print x starting from 0

Ex:
x = 10
for i in range(4):
    x = x + 1
    print(x)
This will print 11, 12, 13, 14

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

```
Output
Loop # 0, x = 10
Loop # 1, x = 11
Loop # 2, x = 12
Loop # 3, x = 13
```

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



10/25/24 LOOPS

to divider

Code

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

Output

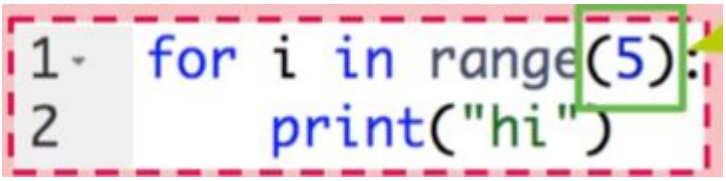
```
i # 0
i # 5
i # 10
i # 15
i # 20
i # 25
i # 30
i # 35
i # 40
i # 45
i # 50
i # 55
i # 60
i # 65
i # 70
i # 75
i # 80
i # 85
i # 90
i # 95
i # 100
```

Going up by 5 starting from 0 to 100

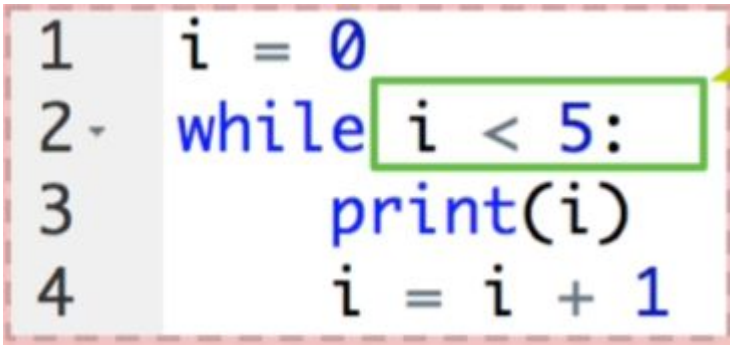
10/29/24 LOOPS

BACK to divider

FOR Loops runa certain number of times defined in range()



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



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))
```

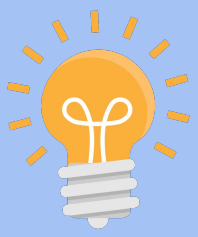
adds 2 + 3 + 4 + 5 and prints:

```
14
```

UNIT 4 EXTRA PAGE

BACK to divider

BACK to cover



Unit 5:

BACK to divider

Links

- EarSketch
- EarSketch Vocabulary
- New Curriculum

NOV 12, 13

BACK to divider

5.0 Marking Period 1 Reflection

What learning strategies did you use this year that made your work better?

What problems did you encounter while working, whether it is bugs or just not understanding the concepts given?

How do you feel about your grade, is it justified for how much work you did? How can you improve for next time?

Do you feel that the way you take notes hinders your understanding of the topic? How can you improve your note taking skills in your notebook?

Reflecting on Peer Feedback and Creating a Plan of Action

Fahim Ahmed

One thing I love about your project is how you included your Digital Journal and others did not. You can improve by adding the flowcharts to your CFUs.

HAILEY SOOKNANAN

The layout out is very nice and manageable to navigate, could improve on the the border color to not be too bright.

Malachi Kennedy

I love the way you did the CFUs. You can improve by adding your flow charts and adding images.

Nafisa Nawshin

I like how you added your digital journal and your projects however you can improve by giving your cfus a proper title

Nibrus Chowdhury

All the required content that is suppose to be there, is there. The UI is a little bit confusing and could use retouching in the future to make it a better experience for the viewer.

MARISA RAMAUTAR

i liked the layout and the colors you used for the portfolio. Next time I think you should make the length of cfus that don't have the flowchart shorter so its easier for the user to use.

AMELIA DAVY

I liked the layout of your pages and the navigation is nice. Although maybe you could make the drop downs shorter for the ones that don't have that much information in them so it's easier to navigate.

FARHAN KAMAL

Great start. Just continue improving on the design (I noticed the empty space below the homepage). Also add some context of the work on the title as well.

SHARENA SHARIF

I feel like the overall aesthetic of your layout could be improved, the color scheme is nice but I think you couldve done more, CSS, wise. Essentially, work on the visual appeal. Otherwise it is consistent. You did a good job on the content, its very thorough, with even your digital notebook included. Its different, and adds more data about you and your work ethic. Navigation wise, the links work, though I would like a way to go back to your page. Maybe make it an Iframe. Anyways, color scheme is nice and I think you could add more CSS components.

SHELDON GEORGE

This is very good, but you should include the names of the CFUs in the portfolio so that the user can have an idea of what they are about.

Arafath Ahmed

I really liked the nice and simple type of design you have on your webpage. Your links also work very well. I suggest to add the flowcharts to the CFU's. The way you made the dropdowns for them looked really nice. Overall, this is a pretty good webpage.

NOV 13

BACK to divider

Reflecting on Peer Feedback and Creating a Plan of Action

Write a Reflection about the feedback you received

Something that surprised me is how many people liked my digital notebook, I thought a lot of people would've added it but barely anyone did

My peers noticed that I had a lot of empty space whether it be in the home page or just any of the dropdown pages

I agree with the fact that I need to do something with the empty space whether it is add more information or I can take it out to make it easier to navigate my page

I feel that the suggestions of updating my UI resonates with me. I want to make a page less basic and boring and add more animation and interactivity.

I disagree with the people saying that I need a title for the CFUs, we did not have a title for any of the CFUs, we only had a number and description, leaving it like that will make it a lot more simple to navigate and find a specific one as I already labeled their units

NOV 13

BACK to divider

Action Plan

Goal: Ensure there is less unused space and easier to navigate the page

Steps:

1. Make home page length shorter
2. Make cfu dropdowns proportional to the content
3. Make projects and labs proportional to content
4. Make digital notebook and earsketch page smaller

Timeline: Finish by 12/1/24

Goal: Make UI more interactive and smooth

Steps:

1. Make Menu dropdown a dropdown and make each clickable link bigger when hovered over.
2. Make each CFU flowchart bigger when hovered over

Timeline: Finish by 2025

Summary Reflection

I received a lot of good feedback, I was told about the empty space I had, I had a consistent but a dull UI that can be hard to navigate because of the long dropdown pages. I will plan to change both my UI and minimize my empty space in my page, I will do this by adding more pictures or kinds of links as images. I will add my sophomore portfolio to my main page so they can access that and see how far I've come. I will first start by fixing my dropdown pages and customize each to how much content is in each, then I will focus on updating my UI as that is where I will be struggling the most.

Doing this will make my junior portfolio better and keep me ahead of the game.

EARSKETCH

to divider

EarSketch code example:

1. from earsketch import *
2. setTempo(#) - 120
3. init( ) initialize
4. Music code
5. Music code
6. Music code
7. Music code
8. finish ( )

Track start end
fitMedia(soundvariable, #, #, #)

Track start
makeBeat(soundvariable, #, #, beatvariable)

Variable

Sound voice = sound
Beat SBeat = “String 0-+”

```
fitMedia(voice, 1, 2, 4)
```

```
snapBeat = “0 - - 0 - - 0”
clapBeat = “+ + 0 0 - - 0 0 + +”
knockBeat = “- 0 - 0 - - - 0 0 0 -”

makeBeat(voice, 2, 1, snapBeat)
```

EARSKETCH UNIT 1

BACK to divider

Chapter 1 Summary

Sound.mp3 file
Video.mp4 file
Gdoc.doc/pdf.txt

- A computer program is a sequence of instructions the computer executes to accomplish a specific task.
- A script is a nickname for a short program.
- A DAW, or Digital Audio Workstation, is a type of computer software for recording and editing audio. EarSketch is a DAW that lets you make music with code.
- To make music, type code into the editor, click "run", and click "play" to listen.
- Sounds can be found in the Sound Browser. You can add sounds to your code by using the sound constant in a function like `fitMedia()`.
- Programming languages are collections of words and symbols that are understood by the computer.
- The rules of syntax define how code must be written and how punctuation (`.`, `,`) is used.
- Create a new script by clicking the large blue link "Click here to create..." or the "+" icon on the editor tabs.
- `fitMedia()` is a way of adding sounds to the DAW. It has the following four parameters.
    - Sound: the sound constant from the sound browser
    - Track: the track number
    - Start: the starting measure
    - End: the ending measure
- Debugging is the process of finding and fixing bugs, or errors, made by the programmer.
- The console shows information about the state of a program, making it useful for debugging syntax errors.
- Common programming errors include typos, incorrect cases, missing parentheses, improper script setup, and logic errors.

Chapter 2 Summary

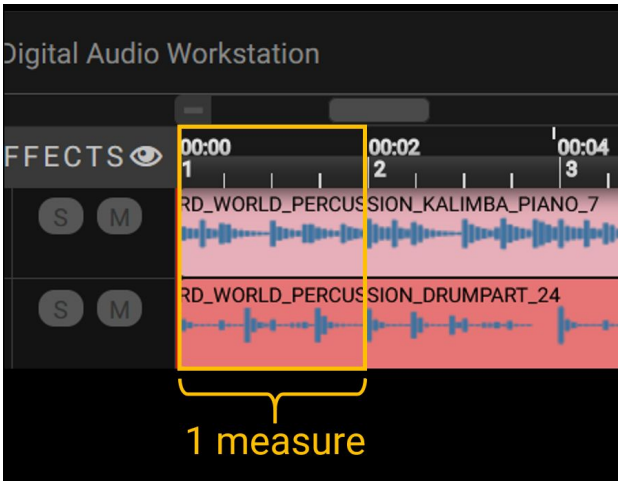
- Tempo is the speed at which a piece of music is played, specified in beats per minute (bpm). Tempo is tied to genre.
- The sounds in the EarSketch Sound Browser are organized into folders of related sounds. To see the exact tempo of a clip, hover over the name in the Sound Browser.
- Comments are lines of code that are ignored by the computer. They are useful for making notes within a script.
- `from earsketch import *` adds the EarSketch API to your project.
- `setTempo()` lets you specify the tempo of your song. It must be included in every EarSketch script.
- You can upload your own sounds to EarSketch through the Sound Browser. Just click "Add Sound".
- A process is a program running on a computer. Processing is carried out by a computer's CPU, which is responsible for executing program instructions.
- Memory (a.k.a RAM or primary storage) holds data and processing instructions temporarily for the CPU to use.
- Secondary storage refers to long term storage of data, often in high volumes. Data from secondary storage must be put into memory before the CPU can access it.
- Copyright is a portion of law that covers ownership of creative works, like music. It is important to musicians because it defines how another person's work can be used and shared.
- If you create a musical work that is tangible and new, you have an automatic copyright. In other words, you have rights over the work you created.
- Licensing a piece of music gives others permission to use it.

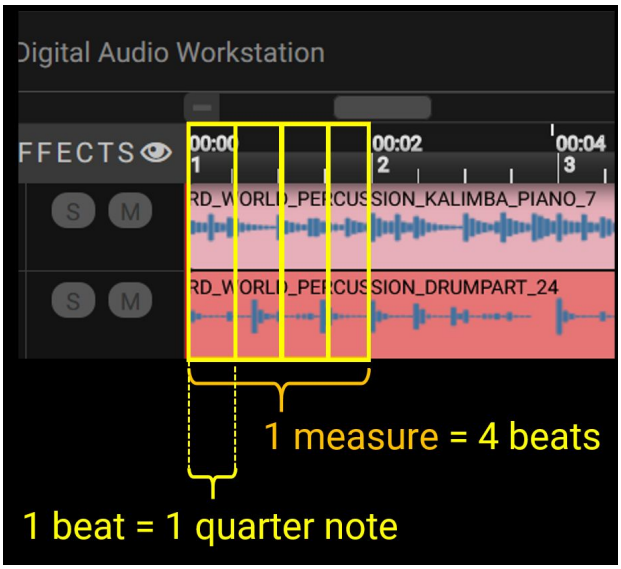
EARSKETCH UNIT 1

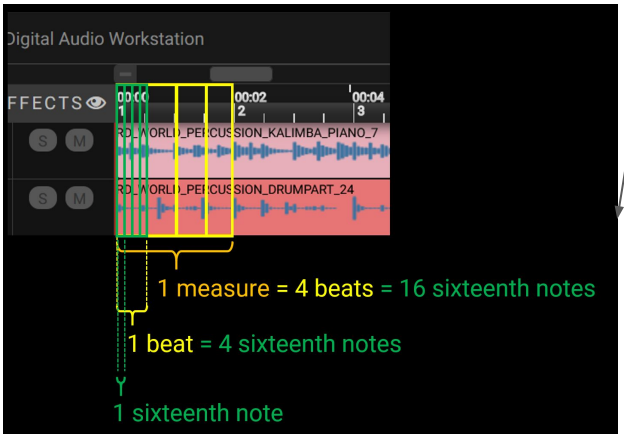
BACK to divider

Copyright is the part of law that covers intellectual property, or ownership of creative work, like music. When using samples (small pieces of music) or remixing existing music, you need to give credit to the authors, and you can do so in the comments of your code. Before using sounds from other musicians and sharing your own music, learn more about copyright!

Copyright infringement is a violation of copyright, like illegally downloading music. In the United States, fair use allows for use of copyrighted content under certain conditions, like educational or critical purposes, reusing only small amounts of the work. Fair use disputes are determined by a judge on a case-by-case basis.







Chapter 3 Summary

- In EarSketch, 1 measure is divided into 4 beats, or 4 quarter notes. 1 beat is divided into 4 sixteenth notes.
- Variables create a space in computer memory to store data.
- You first need to assign the value (data) to the variable name using the sign `=`. Then you can use the variable by writing its name in the code.
- A string is a data type that consists of a series of characters encapsulated by single or double quotes.
- Percussive sounds can be found using the GENRE filter in the Sound Browser: select the artist MAKEBEAT.
- Strings are used with the `makeBeat()` function to create rhythmic patterns in EarSketch. `makeBeat()` takes a beat string to define each sixteenth note of its pattern. A `0` starts playing a sound, a `+` extends the note for the next sub-beat, and `-` creates a rest.
- `makeBeat()` takes four arguments:
  - Sound: A sound constant
  - Track: The track number
  - Start: The starting measure
  - Beat: A string composed of the characters `"0"`, `"+"`, and `"-"`.
- Different beat patterns often correspond to different musical genres.

EARSKETCH UNIT 2

BACK to divider

Beat I made with earsketch

```
# 4.5 Musical tips: Theme is spooky
from earsketch import *
setTempo(100)
#song
#variables
drum1 = OS_SNARE03
beat = HOUSE_BREAKBEAT_013
clap = AK_UNDOG_CLAPS_SNAPS_2
beat_string = "0-0-0-00-000-000"
electro = ELECTRO_ANALOGUE_SPACELEAD_001
bird = HARBANSSINGH_HARBANSBIRD
# Looping our beat
for measure in range(1, 7):
    makeBeat(beat, 1, measure, beat_string)
# Fitting media
fitMedia(clap, 2, 2, 6)
fitMedia(electro, 3, 3, 6)
fitMedia(bird, 4, 1, 2)
finish()
```

analyzeTrack is a function that you can use to check the volume of a track and print it to see

```
loudness1 = analyzeTrack(1, RMS_AMPLITUDE)
print(loudness1)
```

loudness is 0.10306035109110138

Conditionals can be used to change volume or effects of a song based on the volume it already has. It can be good to make sure the main tracks are higher than the other ones.

Chapter 4 Summary

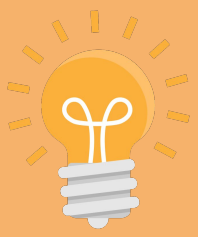
- A `for` loop instructs the computer to execute a code section repeatedly, requiring less code. `for` loops consist of a loop body, loop counter, and range. The code in the loop body must be indented.
- The control flow represents the order in which statements are executed by the computer.
- `print()` displays the data in the console. It is a useful tool for debugging because it allows the programmer to learn the state of the program.
- Printing, commenting out code, and the console can all be used to debug code. Additionally, asking someone for help can significantly speed up the debugging process.
- Revisit the expanded list of common programming errors: Common Errors.
- The pitch of a sound determines how high or low it sounds on a relative scale.
- The key of a song defines the scale, or group of pitches, in which the piece is composed, as well as the tonic note. Keys are either major or minor, which tend to give a different impression to the listener.
- You can use 3 basic tracks for the backbone of your songs: higher pitched melody, lower pitched bass, and percussion.
-

Chapter 5 Summary

- Effects change the qualities of a sound to make them unique.
- Volume is related to loudness. Delay creates an echo. Reverb makes it feel like the sound is played in a large room. Panning places your music on the left or right side.
- Effects are implemented in EarSketch with the `setEffect()` function. Its syntax is `setEffect(track, type, parameter, value)`.
    1. track: Track number or `MIX_TRACK`
    2. type: Effect constant, like `DELAY`
    3. parameter: Parameter constant, like `DELAY_TIME`
    4. value: Parameter value, like `200`. This value must be within the effect's range.
- Functions contain instructions for the computer to execute. Data is sent to functions by arguments, which affect how the function executes. The syntax of a function call with two arguments is `myFunction(argument1, argument2)`. An example of syntax used in a function call with 4 arguments is `makeBeat(kick, 2, measure, kickBeat)`.
- A complete list of EarSketch effects and their parameters can be found in effects, along with descriptions for each.
- Envelopes define how an effect parameter changes over time. They are described with value-time pairs, like (value, time, value, time).
- For an envelope, the 7-parameter `setEffect()` arguments are: `setEffect(track, type, parameter, startValue, start, endValue, end)`.

Chapter 6 Summary

- The `analyzeTrack()` function takes two arguments: the track and feature. When the feature is `RMS_AMPLITUDE`, the function will return the loudness of the track (a number between 0 and 1). When the feature is `SPECTRAL_CENTROID`, the function will return the brightness of the track.
- The boolean data type has only two possible values, `True` and `False`.
- Boolean values are generated by comparison operators: `==`, `!=`, `>`, `>=`, `<`, `<=`.
- `==` evaluates if 2 values are equal, whereas `=` assigns a value to a variable.
- An operator is a character that represents an action.
- Expressions are evaluated by the computer to produce a value.
- A statement is an instruction for the computer.
- A condition is an expression that evaluates to `True` or `False`.
- The `if` statement only executes its code block when its condition is `True`.
- In the event that an `if` statement's condition is `False`, an optional `else` statement allows an alternative code block to be executed.



Unit 6: Graphics

CFM 12/4/24

to divider

- **What do I want my life to be like in 10 years from now?**

  I want to be a businessman, maybe get into finance, I want to also entrepreneur and create my own businesses, I

- **What am I willing to do to get there?**

  I am willing to sacrifice a lot of years and time now to be able to get experience and build up to my main goal with also following my dreams/other goals

- **What colleges offer the type of education I need?**

  Stony Brook University, Baruch

- **Look at the Education ROI site**
- **Determine what colleges and careers are for you?**

  I can go to Adelphi, Stony Brook, or Baruch

- **How much have I save to go to college?**

  Some cost around 17k after financial aid for 1 semester.

- **How much would I have to borrow to afford the college of my choice?**

  If I start working or work during the summer and college I can make a lot of money before I start, I will try to live at home with my parents as that will save me a lot of money and stress.

Unit 6 Review

to divider

```
import simplegui

def draw_handler(canvas):
      canvas.draw_point([x,y], "color")
      canvas.draw line([point1 (x1,y1)], [point2(x2,y2)], line width, color)
      canvas.draw_polygon([(x1,y1)], [(x2,y2)], [(x3,y3)], line_width, color)
      canvas.draw_circle(centerXY, radius, line_width, color)
      These are the 4 functions for drawing in python
      Drawing functions go here

frame = simplegui.create_frame("Dots", 400, 400) height and width of graph
frame.set_canvas_background("White") background color
frame.set_draw_handler(draw_handler) sets the frame with the things from draw_handler
frame.start()
```

Starting a Drawing

the `def draw_handler(canvas)` line tells Python that we are defining a function called draw_handler, and it takes a parameter called canvas. Canvas is a parameter that has all of our drawing functions contained in it. We did not have to do the work of defining canvas ourselves; all of the functions we need are already there within the simplegui module.

The `simplegui.create_frame()` command near the end of the code creates a window titled "Python", with a width of 500 and height of 400. The `frame.set_draw_handler(draw_handler)` line tells the frame that it should constantly be calling that function. On its own, defining our draw function to be **"draw_handler"** has no meaning at all. It could be named anything - the important component of the code is that we tell the frame which function it should call.

Drawing Lines

The `canvas.draw_line()` function has four parameters: a starting point, an ending point, the width of the line, and the color. If we have multiple draw calls (for example, we draw some text and some lines), our lines and text might overlap. The draw function calls that come later in the code will draw on top of what we drew earlier, just like if you were drawing on a piece of paper.

The parameters for this function are as follows:

```
canvas.draw_line(point1, point2, line_width, line_color)
```

The coordinate pairs for our points come in the form of (x, y) or [x, y]. The use of parentheses and square brackets are interchangeable, as long as they match.

Drawing Polygons

In addition to lines, we can draw polygons. The `draw_polygon` function has up to four parameters: a list of points, the line width, the line color, and the fill color. Essentially, this function will draw all the points you specify and then connect them with lines, in the order which you specified. Besides the list of connected points, we also specify the width of the connecting lines and the color of those lines. Then, optionally, we have the ability to fill the inside of our polygon with a color as well.

Here are the generic parameters for this function:

```
canvas.draw_polygon(point_list, line_width, line_color, fill_color)
```

We can draw circles with the `canvas.draw_circle()` command.

```
canvas.draw_circle((x,y), circle_radius, line_thickness, line_color, fill_color)
```

This command takes four parameters, plus one optional parameter. The first is the location of the center of the circle. We also include parameters defining the radius of the circle, the thickness of the line, and the color of the line. The fifth, optional, parameter is the fill color, which lets us color the interior of the circle with a color of our choosing.

```
canvas.draw_circle((150, 180), 30, 10, "Yellow", "Black")
```

There is no built in command to create part of a circle, such as a semicircle or an arc. Instead, the next best option is to cover up the part of the circle we don't want by adding another shape or line.

Unit 6 Review

BACK to divider

Unit 6: Test Review

1. Define the following:

a. global variable - A copy of the variable that used throughout the whole program

b. range function - Creates a list of numbers

c. for loop - A for loop is a loop that runs the code inside of it as many times as you want it to run

2. When do you use a for loop instead of a while loop?

- When there is a starting and ending point,

3. Write a line of code that will draw a solid box from point (108, 67) to (56, 178).

```
canvas.draw_polygon([(108,67),(108,178),(56,178),(56,67)], 5, "purple", "orange")
```

4. Write a line of code that will draw a green circle at the point (78, 119) with a radius of 170.

```
canvas.draw_circle((78,119), 170, 5, "green")
```

5. For a horizontal line the **y** values are the same.

6. Write a FOR loop that will draw 3 circles with a radius of 20 exactly 50 points apart in a vertical line. The first point should be (100,100).

```
for x in range(100, 250, 50):
    canvas.draw_circle((100,x), 20, 5, "black")
```

7. For the following circle statement match each letter with what it does.

canvas.draw_circle ((500, 300), A, B, C, D)

a. fill color D

b. line color C

c. radius A

d. line weight B

8. Write the code to draw a red point at (8, 90)

```
canvas.draw_point((8,90), "red")
```

9. Graph the following points:

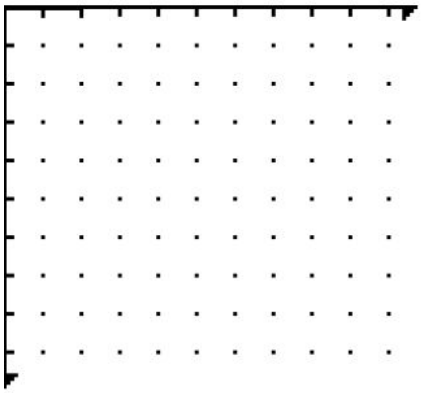
a. (8, 7)

b. (5, 3)

c. (2, 5)

d. (7, 5)

e. (3, 8)



10. What color does each command make?

a. RGB (250, 0, 0) - red

b. RGB (0, 250, 0) - green

c. RGB (250, 250, 0) - yellow

d. RGB (0, 0, 255) - blue

e. RGB (100, 100, 100) - grey

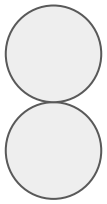
Unit 6 Review

to divider

11. Rewrite the following while loop as a for loop:

```
C = 10
while (c>20):
      C = c+4
      print c

for c in range(14,24,4):
      print(c)
```

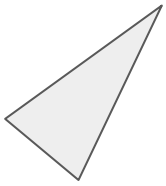


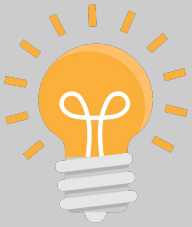
12. What does the following code draw?

```
canvas.draw_circle ((100, 75), 50, 10, "Black")
canvas.draw_circle ((100, 175), 50, 10, "Black")
```

13. What does the following code draw?

```
canvas.draw_polygon([(350, 200), (200, 450), (50, 350)], 5, "Black", "Gray")
```



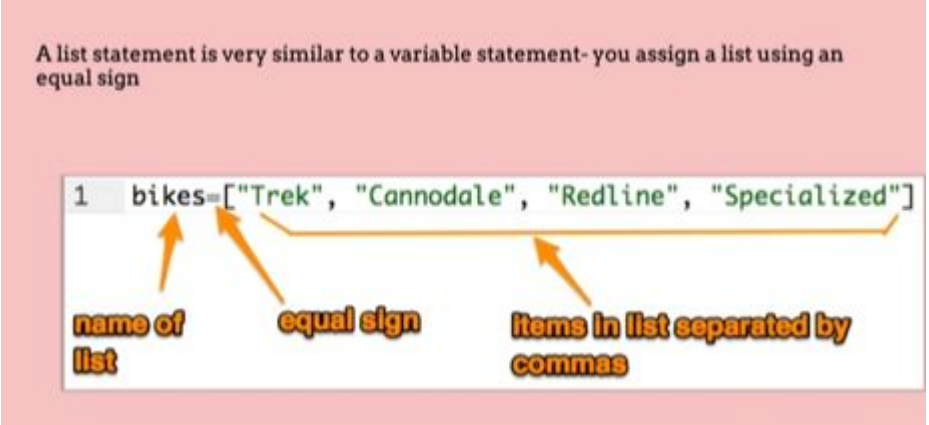


Unit 7: Functions

In the years that I started doing python, we went over a lot of units. We started with getting used to using variables and learning what printing, strings. We then learned how to do number calculations and being able to store and output that data. Then we learned how to do conditionals and let our code make a decision based on user input. We then learned how to use loops and repeat things in our code. Then we learned how to code music. Lastly, we learned how to create UI with graphics. I feel my grades are a pretty good reflection of how I am doing so far. I feel I am making progress on par with my class. I feel that I am strong in coding on my own but I can get lazy and struggle working with others. My goals for the remaining part of the semester is to keep my grades up and make stuff up. I want to start studying 30 minutes and try to make it fun by giving myself tasks. I plan to use my UI ideas

LIST REVIEW 12/19/24

to divider



A **list** is a collection of items that have a particular order

- Since a list usually contains more than one item, it is a good idea to name the list with plural names like: items, digits or letters
- In Python, square brackets [] indicate a list
- Each item in a list is separated by commas
- A list can have multiple data types (e.g. int, string, etc) within a single list

integer function
1 3

favorites = ["Food", int, float, function(), "other"]

String decimal
0 2

https://py3.codeskulptor.org/#user309_RjpWRMT65xrU6OT.py
This is a list of names of whos bringing what for our party

Unit 7 Review 1/2/25

to divider

Unit 7 Review

Functions are sections of code that are separated and defined so they can be called at a later time and multiple times, using functions makes it easier on the coder by using up less storage and not needing to write the code over and over.

To call a function you need to first define it like this:

def function( ):

This is how to call it:

function( )

This will allow you to name the function and once you click enter you are indented and all the indented information is apart of the function

```
def function( ):
    print("Hello")
```

When using functions you might wnat to save some information that you got from it, you can do this by **return**:

```
def function(x,y):
    z  = x+y
    return z
```

Unit 7 Review 1/2/25

to divider

Unit 7 Review

If you want to have the values the parameters that you use optional, you can do this by making the parameter equal to a number or string:

def function(x,y,z = 2):

Doing this can give you more freedom with how you use functions and rely on user input, a person isn't required to do everything and is already set

Some useful functions that are built into python

print( )
int( )
float( )
abs( )
min( )
max( )
input( )
range( )
while( )
for i in range( )

Unit 7 Review 1/2/25

BACK to divider

Unit 7: Test Review
Define the following for questions 1-3:
1. Subprogram (aka function): **a collection of commands as a block of code that can be called later on**

2. Return: **keyword that tells program to send a value back to a variable**

3. Parameter: **a variable that only exists in the function**

Answer the following for questions 4-8:
4. What are two reasons for using subprograms (alias functions)?
Be more efficient and be more organized
5. What are the steps to add a function to a program?
def the function then call it after
6. What is the difference between a parameter and a function?
A parameter only exists in the function and the function exists in the whole code (
7. Does the following code return a string or a number?

```
def f():
return "3"
```

It will return a string because 3 is inside quotes

8. Why is it necessary to do something (i.e. store in a variable or print) to the value returned by a function?
It is necessary to do something to a variable returned by a function since it will be lost if not stored or printed as it only exists in the function unless something done

Unit 7: Test Review

Write the code for questions 9-10:

9. Write a function to return the larger of two numbers entered from two user inputted values, where the user
inputs are entered after the displays of “First Entry = ” and “Second Entry = ”. The numbers should be
decimal values (not just integers).

```
def greater( ):
    first = float(input(“First Entry: “))
    second = float(input(“Second Entry: “))
    if first > second:
        return first
    elif second > first
        return second
    else:
        print(“they are equal”)
```

10. Write a function to return the word that is first alphabetically from two user inputted text entries, where the
user inputs that text by entering their own words after the displays of “First Entry = “ and “Second Entry =
“. Remember that you can use the < and > operators with strings. (You can assume that the user only
inputs lower case words.)

```
def larger( ):
    first = input(“First Entry = ”)
    second = input(“Second Entry = ”)
    if first > second:
        return first
    elif second > first
        return second
    else:
        print(“they are equal”)
```

Unit 8: Lists

Review from Unit 7:

A **list** is a collection of items that have a particular order

- Since a list usually contains more than one item, it is a good idea to name the list with plural names like: items, digits or letters
- In Python, square brackets [] indicate a list
- Each item in a list is separated by commas
- A list can have multiple data types (e.g. int, string, etc) within a single list

https://py3.codeskulptor.org/#user309_RjpWRMT65xrU6OT.py
This is a list of names of whos bringing what for our party

```
bikes = [“Trek”, “Cannodale”, “Redline”, “Specialized”]
```

List name

Items in list

The **len()** function will find the number of elements within a list. This is the same function as the string len() function

numbers = [1,4,3,6,2]
print(len(numbers))

Output -> 5

The **append()** method will add an element to the end of the list.

```
numbers = [1,4,3,6,2]
print("before:")
print(numbers)
print("length: {}".format(len(numbers)))

numbers.append(99)

print("after:")
print(numbers)
print("length: {}".format(len(numbers)))
```

```
before:
[1, 4, 3, 6, 2]
length: 5
after:
[1, 4, 3, 6, 2, 99]
length: 6
```

To add multiple items you need to append it multiple times which you can do through a for loop

Something about the append is that you need to call it directly on the list with a period while you call the len() function with the list in the parentheses

The **insert()** method will add an element at a specific index. It takes two arguments: the first being an int representing the index, and the second being the value to insert

```
numbers = [1,4,3,6,2]
print("before:")
print(numbers)

numbers.insert(2,99)

print("after:")
print(numbers)
```

Number of place you want to insert it in

Item you want to insert

INDICES:
0 1 2 3 4 5

```
before:
[1, 4, 3, 6, 2]
after:
[1, 4, 99, 3, 6, 2]
```

Accessing Elements in the List

Each elements (or item) in the list has an Index or position. These positions start counting at 0

groceries = [“Eggs”, “Apples”, “Milk”, “Bread”]

- **len()** counts how many items is in a list
- **.append()** adds a new thing to the end of the list
- **.insert()** inserts new item into list where you want it
- **.remove()** the item that was in the list
- **.sort()** organizes the list from least to greatest
- **.lower()** makes the item lowercase
- **.upper** makes the item uppercase

Here is code of how I used the list
https://py3.codeskulptor.org/#user310_baLC4onYQJ7FaRI.py