Coding Python

What is Python?

Python is an interpreted object-oriented programming language. Unlike HTML, it cannot be run natively on many devices without being installed. Luckily, there are many Python compilers around the internet, so you can run it just with a browser and an internet connection. Python also has many libraries that you can use to program in new ways! What will **you** make with this power?

Table of Contents Unit 1 - The Basics (Page 1)

Program 1 - Greet by Name

What is Python?

Lesson 1.1.1

Lesson 1.1.2

Lesson 1.1.3

Lesson 1.1.4

Program 2 - Calculator

Lesson 1.2.1

Lesson 1.2.2

Lesson 1.2.3

Lesson 1.2.4

Lesson 1.2.5

Turn the page

Table of Contents Unit 1 - The Basics (Page 2)

Program 3 - Functions

Lesson 1.3.1

Lesson 1.3.2

Lesson 1.3.3

Lesson 1.3.4

Unit 1 Test (Unavailable via OpenEducation)

Turn the page to Unit 2

Table of Contents Unit 2 - Diving Deep (Page 1)

Program 1 - Arrays/Lists

Lesson 2.1.1

Lesson 2.1.2

Lesson 2.1.3

Lesson 2.1.4

Lesson 2.1.5

Program 2 - While and For loops

Lesson 2.2.1

Lesson 2.2.2

Lesson 1.1.1

Goal

I can use the basic functionalities of a variable and printing text to the console.

Key Scripts

```
print(data)
```

```
varname = data
```

Analyze the code. What do you think it does?

```
print("Hello, World!")
```

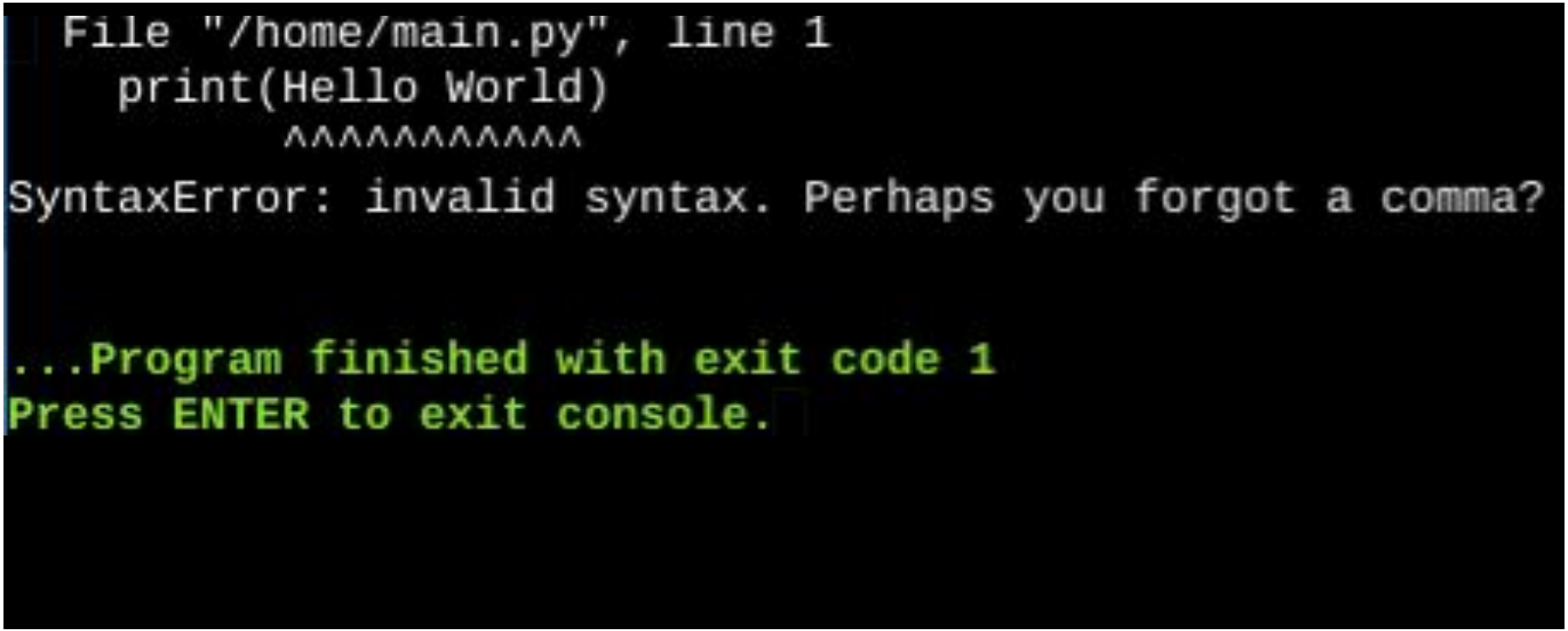
This is the output.

```
Hello, World!
```

What if you don’t use quotes?

```
print(Hello, World!)
```

This is the output.



Why?

Quotes make code into *strings*. Strings are basically just text. (*Strings* of letters.) If you do not use quotes, your text could be parsed as a variable, function, or line of code. Remember to include them!

Let’s make a variable!

Like in math, variables store information. You do not need to put quotes around the variable **name**, only the **data**. Here is an example:

```
x = "Hello!"
```

To make or modify a variable, all you need to do is:

```
varname = data
```

That’s it!

Printing variables

A variable *name* is **not** a string. Variables may *contain* strings, or even **other** data types. This means you do **not** need to put quotes around a variable *name*. This is how you display a variable on the screen.

```
print(varname)
```

Let’s make a variable called x. It has the value "Hello, World!" We will then print it.

```
x = "Hello, World"

print(x)
```

This is the output.

```
Hello, World!
```

REMEMBER!

Variables don’t just have to be called x, y, or z. You can name them mostly anything!

Exceptions:

- A name that is just numbers
- Specific keywords
- Anything including parenthesis "()"
- Anything containing a "."
- Anything with spaces
- SO MANY MORE!

Now, make your own programs!

This is required. Here are some ways you can do it! Make sure to save the file! Make sure to save the file!

- If you know how, you can make them on your calculator.
- Otherwise you can make them on the OnlineGDB Python Compiler!

Lesson 1.1.2

Goal

I can use the basic functionalities of an input.

Key Scripts

```
input()
```

Inputs

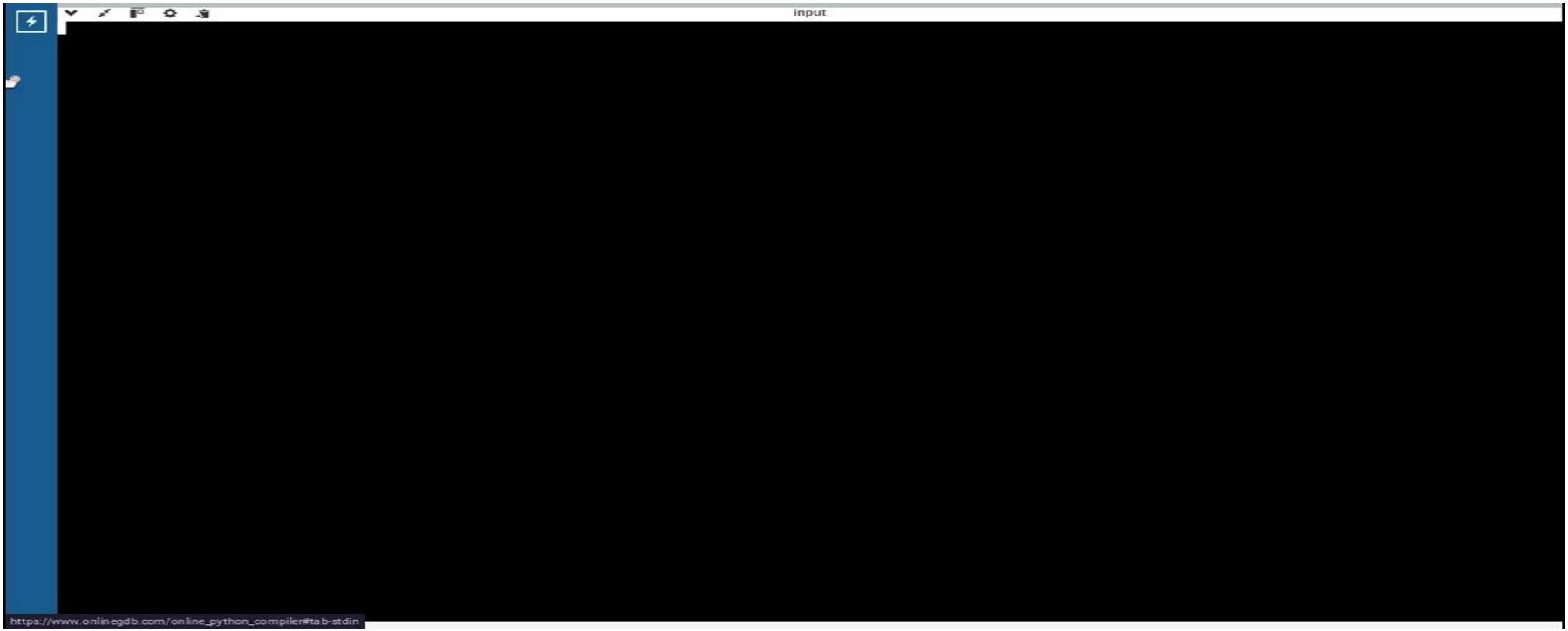
Now, making variables and printing is great and all, but what about actually *interacting* with the code? Let's set a variable to some inputted text.

```
x = input()
```

Now, we can print x.

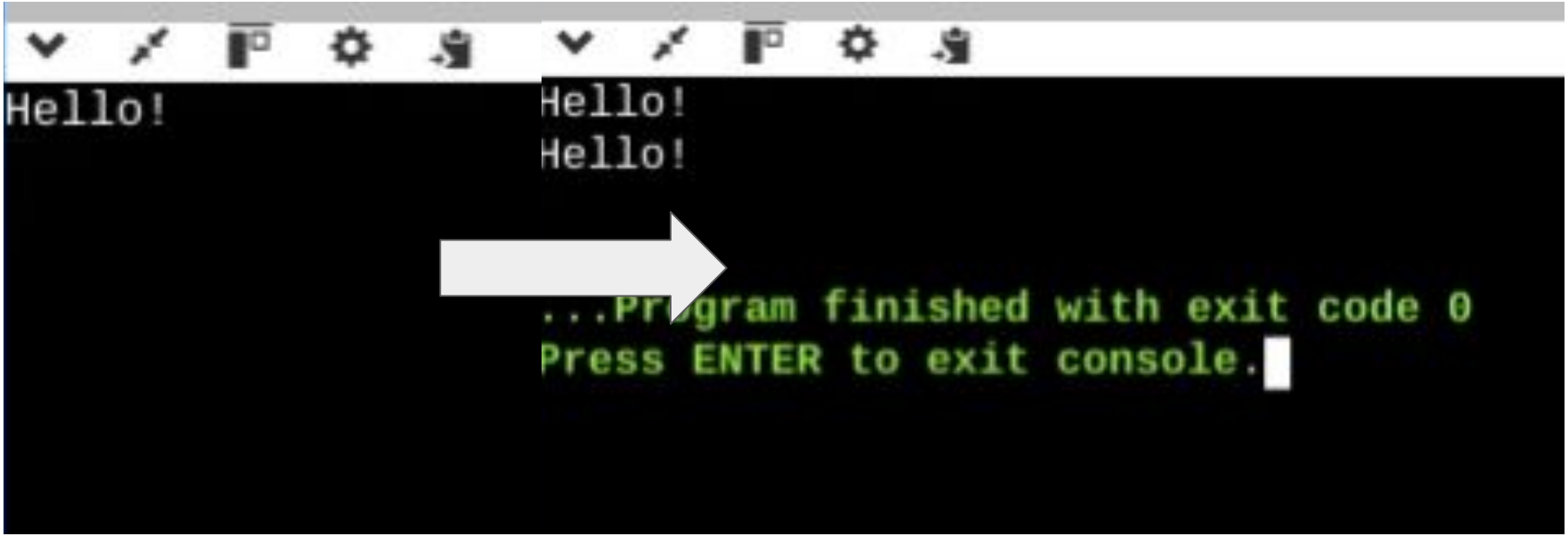
```
print(x)
```

This is the output.



What did it do?

I typed in "Hello!" and clicked enter. It printed it out back to me.



Now, make your own programs!

This is required. Here are some ways you can do it! Make sure to save the file!

- If you know how, you can make them on your calculator.
- Otherwise you can make them on the OnlineGDB Python Compiler!

Lesson 1.1.3

Goal

I can add integers, concatenate strings, and set strings to integers.

Key Scripts

`int(`*`data`*`)`

`data + data`

Adding numbers

To add numbers together, just use +!

```
print(1 + 1)
```

This is the output.

```
2
```

Concatenating Strings

To add strings together, just use + again!

```
print("Skibidi " + "Rizzler")
```

This is the output.

```
Skibidi Rizzler
```

Strings as numbers

Inputs in Python are *always* parsed as strings (text.) If you do a string + a number, it throws an error. To turn a string, i.e. "123" into a number, use this code.

```
int("123")
```

Let’s print it out. Let’s also print it without the "int()".

```
print("123"+"456")
```

```
print(int("123")+int("456"))
```

This is the output.

```
123456

579
```

It works!

The first one is just 123456 and the second one is the answer to the addition problem 123+456!

Inputs as numbers

Remember, Inputs in (some types of) Python are *always* parsed as strings (text.)
Now, let's make a simple calculator that adds 1 to the input.

```
x = input()

print(x + 1)
```

This is the output.

```
(typed in) 11

12
```

Now, program an addition calculator.

Use your knowledge of inputs, addition, and integers to do this.

This is required. Here are some ways you can do it! Make sure to save the file!

- If you know how, you can make it on your calculator.
- Otherwise you can make it on the OnlineGDB Python Compiler!

Lesson 1.1.4

Greet by Name

Make a program where you can type in your name, and then it greets you by name. Please make this on OnlineGDB Python Compiler!

See next slide

Example output:

```
(typed in) Jade

Hello, Jade!
```

Lesson 1.2.1

Goal

I can add, subtract, multiply, divide and use exponents in python.

Key Scripts

```
data + data

data - data

data * data

data / data

data ** data

int(data)

#comment
```

Operations

```
print(3 + 3) #Addition

print(3 - 3) #Subtraction

print(3 * 3) #Multiplication

print(3 / 3) #Division

print(3 ** 3) #Exponents
```

This is the output.

```
6

0

9

1.0

27
```

Why the decimal?

When you add, subtract, multiply, or use exponents on two integers, it is GUARANTEED that the output will be an integer. With division on the other hand, you can get a decimal output. So, python parses operations done with two ints as an int UNLESS it is a division (or some other ones, too) problem. Numbers with decimals are parsed as **floats** or floating points. To counteract this, if the number has no decimals, let’s just turn it into an int!

New program!

```
print(3 + 3) #Addition

print(3 - 3) #Subtraction

print(3 * 3) #Multiplication

print(int(3 / 3)) #Division

print(3 ** 3) #Exponents
```

This is the output.

```
6

0

9

1

27
```

#commentsslay #100% #lol

You might've seen the hashtags. These make comments in your code. Once you write a hashtag, the entire line after that becomes a comment. Programs you write in comments won’t run.

```
# This is a comment!

# print("This won't print!")
```

This is the output.



Now, make your own programs!

This is required. Here are some ways you can do it! Make sure to save the file!

- If you know how, you can make them on your calculator.
- Otherwise you can make them on the OnlineGDB Python Compiler!

Lesson 1.2.2

Goal

I can make and use an if-then-else loop.

Key Scripts

```
if data:

else data:

data == data
```

If-Then

To make and if-then loop, just use:

```
if condition:

 What to do
```

Make sure to **always** use an indentation. This tells Python that the indented area is part of the loop. The indentation across the entire inside (unless another indented area must be put in) **must be indented the same amount.**

Do this!

```
if condition:
 print("Happy")
 print("Happy")
 print("Happy")
```

Not this!

```
if condition:
print("Happy")
print("Happy")
print("Happy")

if condition:
  print("Happy")
 print("Happy")
   print("Happy")
```

x == y

How do you check if two values are equal? You might think to do it using "=", but we already use that to define variables!

```
varname = value
```

Instead, use:

```
val1 == val2
```

Do this!

```
if x == y:
 print("Happy")
```

Not this!

```
if x = y:
 print("Happy")
```

Let’s make some code!

Remember your colons (this thing! -> :) !

```
if x == y:

 print("x=y")
```

Let’s make x and y be the same value!

```
x = 1

y = 1

if x == y:

 print("x=y")
```

This is the output.

```
x=y
```

Let’s make x and y be the different values!

```
x = 1

x = 2

if x == y:

 print("x=y")
```

This is the output.



Let’s make a new script!

This one will tell you whether or not x = y.

```
if x == y:
 print("x=y")
else:
 print("x does not equal y")
```

Let’s use this.

```
x = 0

y = 0

if x == y:

 print("x=y")

else:

 print("x does not equal y")
```

This is the output.

```
x=y
```

Let’s now use this.

```
x = 0

y = 1

if x == y:

 print("x=y")

else:

 print("x does not equal y")
```

This is the output.

```
x does not equal y
```

Now, make your own programs!

This is required. Here are some ways you can do it! Make sure to save the file!

- If you know how, you can make them on your calculator.
- Otherwise you can make them on the OnlineGDB Python Compiler!

Lesson 1.2.3

Goal

I can make and use an if-then-elif loop.

Key Scripts

```
elif data:
```

```
not data
```

Elif (Else if)

Elif in python basically is else, but on a condition. The two below programs do the same thing.

Program 1:

```
else:
 if x == y:
  print("x=y")
```

Program 2:

```
elif x == y:
 print("x=y")
```

Not

The lowercase word "not" means not. (BIG BRAIN) It makes a boolean (True or False) be the opposite. So,

```
not (x == y)
```

Is true if x does **not** equal y.

This is the program from lesson 1.2.2

```
if x == y:
 print("x=y")
else:
 print("x does not equal y")
```

This is the program if we make the else into elif.

```
if x == y:
 print("x=y")
elif not (x == y):
 print("x does not equal y")
```

Now, let’s just define x and y!

```
x = 0

y = 0

if x == y:

 print("x=y")

elif not (x == y):

 print("x does not equal y")
```

This is the output.

```
x=y
```

Another one:

```
x = 0

y = 1

if x == y:

 print("x=y")

elif not (x == y):

 print("x does not equal y")
```

This is the output.

```
x does not equal y
```

Now, make your own programs!

This is required. Here are some ways you can do it! Make sure to save the file!

- If you know how, you can make them on your calculator.
- Otherwise you can make them on the OnlineGDB Python Compiler!

Lesson 1.2.4

Goal

I can prepare to program a calculator with addition, subtraction, multiplication, and division.

Starting point

Let’s start with just collecting the data to use. We will do this with inputs. x and y need to be integers. The operator is a string.

```
x = int(input("x value"))

y = int(input("y value"))

op = input("Operator")
```

Adding addition

Now, we check if the operator is "+". If so, we add the numbers together and print it out.

```
x = int(input("x value"))

y = int(input("y value"))

op = input("Operator")

if op == "+":

 print(x+y)
```

This is the output.

```
x value1

y value2

Operator+

3
```

Lesson 1.2.5

Goal

I can program a calculator with addition, subtraction, multiplication, and division.

Now, make a calculator with all the operations!

This is required. Here are the other requirements!

1. Use addition, subtraction, multiplication, and division. (+, -, *, /)
2. If using OnlineGDB, start from the framework in Slide 79.

Lesson 1.3.1

Goal

I can use the basic functionalities of a function.

Key Scripts

```
def name():
```

What is a function?

A function is like "My Blocks" in Scratch. You have a short string of code (the block) that then runs a longer program. The format for functions is like an if-then. Use a colon and an indent. Always include two parentheses at the end of a function name. After the name, put a colon.

```
def name():

 #What to do!
```

Calling functions

To call a function (run the code) just use the function name. Remember to use the "()". Do **not** use the colon ":" when calling the function.

```
name()
```

Our first function

Let’s make a function that just prints "hi!"

```
def hi():
 print("hi!")
```

Don’t forget to call the function!

```
hi()
```

This is the output.

```
hi!
```

Lesson 1.3.2

Goal

I can use the basic functionalities of function parameters.

Key Scripts

```
def name(params):
```

Params

Function parameters (params) are basically inputs into the function. You define them when you run a function. This is the code to add params when you create a function.

```
def name(params):

 #happyhappyhappy
```

Params (continued)

Params act like variables when in a function. Let’s make our own print function!

```
def text(data):

 print(data)
```

Let’s run it with the data "Hello, World!"

```
text("Hello, World!")
```

Params (continued)

This is the final program:

```
def text(data):
 print(data)
text("Hello, World!")
```

This is the output.

```
Hello, World!
```

Lesson 1.3.3

Goal

I can use the basic functionalities of returning data in a function.

Key Scripts

```
return data
```

Return

Sometimes, functions can be used as variables or data. A built-in function like this is len(). This *returns* the length of a string or array. Let's use it.

```
print(len("Hello"))
```

This is the output.

```
5
```

Return part 2

Success! "Hello" has 5 letters in it! The coolest thing is, we can make our own functions like this!

```
def returnfive():

 return 5

print(returnfive())
```

Make sure to put () at the end!

This is the output.

```
5
```

Return + Params = ???

Let’s tie in params by making an addition calculator function! Make sure you understand the function before continuing.

```
def add(x,y):

 return x+y
```

Let’s make this into runnable code!

Make sure you understand the function before continuing.

```
def add(x,y):
 return x+y
print(add(1,1))
```

This is the output.

```
2
```

It works?

YES! 1+1=2! It works!

Lesson 1.3.4

Goal

I can make programs using functions, returns, and params.

So, do it!

Make some programs! The sky's the limit!

Lesson 2.1.1

Goal

I can create an array.

Key Scripts

```
[data, data, ...]
```

How programs count

Count to 10. Most people say 1,2,3,4,5,6,7,8,9,10. Most people are not programmers. Programming languages start counting from 0. So the first item in a list is actually item number 0. This will soon become very important.

Making a list

A list or array is basically a list of items. The items can be in any data types. The format is:

```
[item1,item,item3...]
```

Data types in lists

These lists are all accurate. They can contain all data types. Lists also count as data types, so they can contain each other.

```
[1,0.23,"hello!"]

[var1,"hi",[1,2,3]]

[0,1,2,3,4,5,6,7,8,9,10]
```

A variable that’s a list

Since lists count as data types, you can make variables be lists.

Welp, you know what time it is!

Create your own array. :)

Lesson 2.1.2

Goal

I can use the basic functionalities of an array.

Key Scripts

```
name[item]

name.index(data)
```

How to read an array

REMEMBER: Arrays and counting always start at 0! The 0th element is actually the first.

To read an array, just use *name*[*item*]. Let's do a quick example.

```
array = ["a","b","c","d"]

print(array[1])
```

This is the output.

```
b
```

Index

If we want to find… say… the index of "c" in the array, how would we do that? With the index method.

```
array = ["a","b","c","d"]

print(array.index("c"))
```

This is the output.

```
2
```

It works!

Remember that the array starts counting at 0. Element number 2 is actually the third element. IT WORKS PERFECTLY!!! (Output is an integer)

Make a program

Be creative! Use a compiler! (This is requried)

Lesson 2.1.3

Goal

I can use a basic understanding to modify the values of an array.

Key Scripts

```
len(array)

name[item] = data
```

Length

This is a quick one. To get the length of an array, (This starts at 1 and is an int) use the len() function.

```
array = ["a","b","c","d"]

print(len(array))
```

This is the output.

```
4
```

It works!

Yay :)

Setting an element

If you want to change an element, just set it as if it were a variable.

```
array = ["a","b","c","d"]

array[2] = 3

print(array)
```

This is the output.

```
["a","b",3,"d"]
```

Lesson 2.1.4

Goal

I can use a basic understanding to modify the values of an array. (Part 2)

Key Scripts

```
array.pop(position)
```

```
array.insert(position, data)
```

Pop

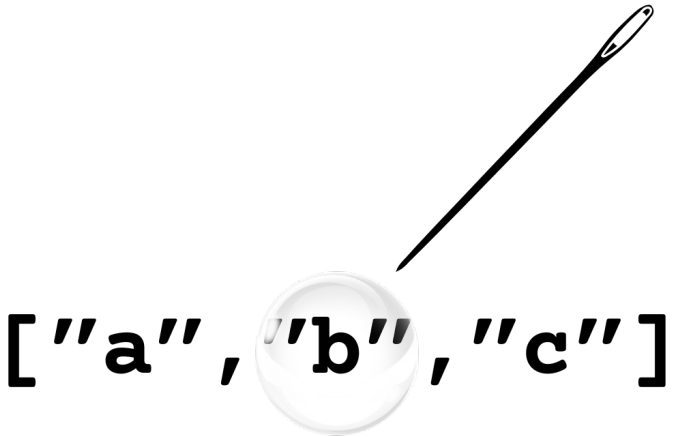
The pop function removes the specified element from an array.

Example:

```
x = ["a","b","c","d"]

print(x.pop(1))
```





This is the output.

```
["a","c","d"]
```

Insert

That worked really well, so now let’s try to do the exact opposite of it! (These are really bad transitions) It’s insert!

```
x = ["a","c","d"]

x.insert(1,"b")

print(x)
```

This is the output.

```
["a","b","c","d"]
```

Lesson 2.1.5

Goal

I can find the index of elements in lists and remove items from lists.

Key Scripts

```
array.index(element)
```

```
array.remove(element)
```

Index

Let’s say you want to know what number element some data is. You would use the index method!

```
x = ["a","b","c","d"]

print(x.index("b"))
```

This is the output.

```
1
```

Remove

Let’s now try **deleting** things from the list! Instead of searching by **index**, we’ll search by the actual **data**!

```
x = ["a","b","c","d"]

print(x.remove("b"))
```

This is the output.

```
["a","c","d"]
```

Try making some programs of your own! Practice makes perfect!

Lesson 2.2.1

Goal

I can use the basic functionalities of a while loop.

Key Scripts

```
while condition:
```

While loops

A **while loop** repeats an action until a boolean is false. (Remember, a boolean is just true or false) So, let's make something run forever! Since true will never be false, let's just make it while true! This is kinda how it would look:

```
while True:

 print("Inappropriate Thing")
```

Notice that it has an indent like a function or if-then.

This is the output.

```
Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing

Inappropriate Thing
```

Try to see what this code does! Use a compiler if you need it!

```
print("type things in and just hit enter sometimes, and try
it out!")

x = ""

while x == "":

 x = input()

print("Program Finished.")
```

Lesson 2.2.2

Goal

I can use the basic functionalities of a for loop.

Key Scripts

```
for x in var:
```

Let’s see what this code does!

```
letters = ["a", "b", "c", "d"]

for x in letters:

 print(x)
```

This is the output.

```
a

b

c

d
```

What happened?

For every item in the list, the for loop ran once, and it set x to the current item. Then, it printed x, the current item. Just like with `len()`, though, it works with strings as it does with lists.

```
for x in "sigma":

 print(x)
```

This is the output.

```
s

i

g

m

a
```

Program Creation

Make a program using a for loop that creates the output shown on the next slide.
For more information, visit slide 157

It should create this output:

```
Coding

with

Pineapple

is

amazing!
```

Now, make the program!

This is required. Here are some ways you can do it! Make sure to save the file!

- If you know how, you can make them on your calculator.
- Otherwise you can make them on the OnlineGDB Python Compiler!

Expert

You are done!

You are now a Python expert. Do your own research, make your own programs, and practice! We will try to make more and more lessons, but it is time for you to learn on your own! Here are some good sources for your coding questions and research.

- Google
- W3schools
- GeeksForGeeks
- Stack Overflow