I. FOUNDATIONAL PROGRAMMING CONCEPTS (INTRODUCTION AND REVIEW)

INTRODUCTION TO THE ALGORITHM

- ***Described in different ways:***
  - Well-defined procedure that allows an agent to solve a problem
  - Clear and unambiguous specification of steps needed to solve a problem
  - Precise "recipe" of solving a problem, so there must be "ingredients" and steps to solve the problem
  - Procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed; agent is a computer or a robot

- ***An algorithm must:***
  - Be well-ordered and unambiguous;
  - Have every operation effectively computable; and
  - Terminate

ELEMENTS OF AN ALGORITHM

- ***Variables***
  - Named memory location that can store a value
  - Box into which one can store and value, and from which one can retrieve a value
  - Only one value is meant to be stored by box
  - Type of value to be placed on the box usually depends on the size of the memory
- ***Operations***
  - Action that allows the manipulation of one or more variables
  - Allow for the program to fulfill its goals and end the program
  - Types:
    - Primitive (input, assignment, output)
    - Conditional
    - Looping

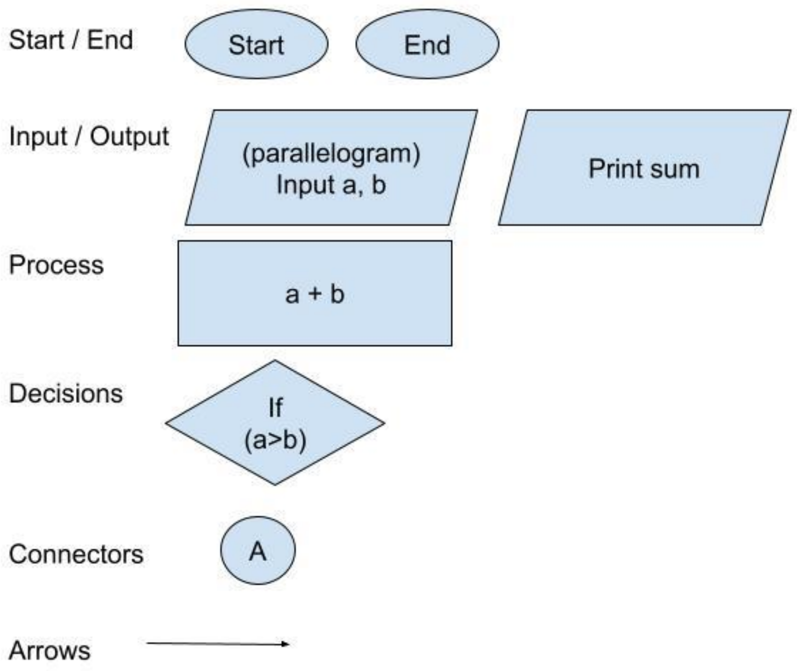
PROBLEM ANALYSIS – CREATING ALGORITHMS

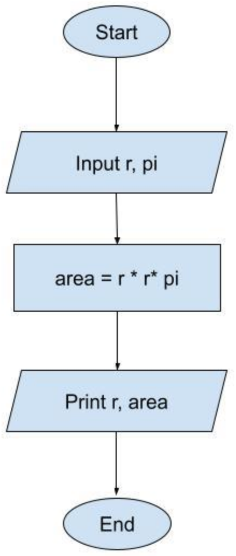
- **Problem:** The radius of a circle is equal to one unit. Compute the corresponding area of a circle and print out the value of the area.
- **Problem Analysis:**
  - Input: radius, pi
  - Process: area= radius*radius*pi
  - Output: area

EXPRESSING ALGORITHMS

- **NATURAL LANGUAGE**
  - Language that a human would be able to read, write, speak, and understand
  - Whatever language rules that the language has is followed in the creation of the algorithms
  - Characteristics:
    - Most understandable, regardless of the IT level
    - Allows for the detail necessary for an algorithm
    - Can also be quite vague at times
    - Humans typically use their natural language which at times can be too rich, ambiguous, and will depend on the language
  - Example:
    - Define the value of radius r and pi (3.1416)
    - Calculate the area of the circle, area = r *r * pi
    - Print out the radius and the computed area
- **PROGRAMMING LANGUAGE**
  - Comprises of words, statements, and rules to produce different types of machine output
  - Construction is very similar to a natural language but has lesser ambiguity
  - Characteristics:
    - Not recommended due to how rigid the rules of a programming language is
    - When solving a problem, it is best to be able to think at a certain abstract level so that more options can be explored

        - Shifts the emphasis of how to solve the problem to tedious details of syntax and grammar
- **PSEUDOCODE**
    - Algorithm in a natural language, but structured to look more like a programming language
    - Artificial and informal language that helps programmers develop algorithms
    - Characteristics:
        - Simple, readable, has very little rules, and doesn't worry much about details like punctuation
        - Allows one to think at an abstract level about the problem
        - Contains only instructions that have a well-defined structure and resemble programming languages
    - Example:
        - NUMBER r, area.
        - INPUT r.
        - area=3.14*r*r.
        - OUTPUT area.
- **FLOWCHARTING**
    - ***Flowchart***
        - Picture of the separate steps of a process which is usually presented in a sequential order
        - Generic type of tool that can visually express an algorithm
    - ***Flowcharting***
        - Process of creating the flowchart
        - Can also include the problem analysis





SAMPLE FLOWCHART:
Compute for the value of
area of a circle using r and pi.

II. BASIC PROGRAMMING

WHAT IS PYTHON?

- It is an interpreted, object-oriented, high-level programming language with dynamic semantics.

"interpreted"

- "It is where the source code is not directly translated by the target machine"

"object-oriented"

- "It is a computer programming model that organizes software design around data, or objects, rather than functions and logic."

"high-level programming language"

- "It is any programming language that is user-friendly for programming and is generally independent of the computer's hardware architecture"

“dynamic semantics”

- "It is an approach to defining where pieces of text are viewed as instructions to update an existing context, the result of which is an updated context."

PRINT COMMAND

- Syntax:
  - print (<add something here>)
  - print (‘hello world’)
  - output: hello world

VARIABLES

- Variables are names that can be assigned a value and then used to refer to that value throughout your code
- Declaring/Summoning/Creating/Using/Syntax:
- <variable name> = <value>
- student_id = 123456

RULES OF VARIABLE NAMING

1. Can be as long or as short as you like
2. May contain uppercase and lowercase letters
3. (A – Z, a – z), digits (0 – 9), and underscores (__)
4. Start with a letter or the underscore character
5. Variable names are case-sensitive (must be
6. named or referred to in the identical fashion)
7. Use very descriptive names

VARIABLE NAMING CONVENTIONS

- Camel Case naming convention
  - myAge
  - favColorInTheColorWheel
- Pascal Case naming convention
  - MyAge
  - FavColorInTheColorWheel
- Snake Case naming convention
  - my_age
  - fav_color_in_the_color_wheel

VARIABLE DATA TYPES

- Are the things/values that a variable can have:
- These are:
  - Integer
  - Floating Point
  - String
  - Boolean
  - Null

INTEGERS

- These are zero, positive, negative, whole numbers.
- There is no explicitly defined limit in the value of the integer Python.
- student_id = 123456

FLOAT

- This represents a floating-point number.
- They are represented with a decimal point.
  - x = 9.9999999999999
- You can also make use of negative values for the floating-point number.
  - x = - 73.435
- student_id = 1.2345

STRING

- Strings are a sequence of bytes representing Unicode characters.

- There are several ways to create a string.
- They differ based on the delimiters and wether a string is single or multiline.
- student_id = ‘meing123’
- **Concatenation** (“+”)
    - Can be used to stitch together strings and values

STRING DELIMITTER

- Can use double quotes ("<any>") or single ('<any>') quotes
    - Is useful for creating sentences containing single quotes
    - To get around this, the "\" can be used.
- For longer strings, you can use the triple quotes ("'<any>'")

BOOLEAN

- Boolean data types determine the truth value of expressions.
- They can either be True or False.
    - booleanVal = True
    - print(booleanVal) # Outputs True

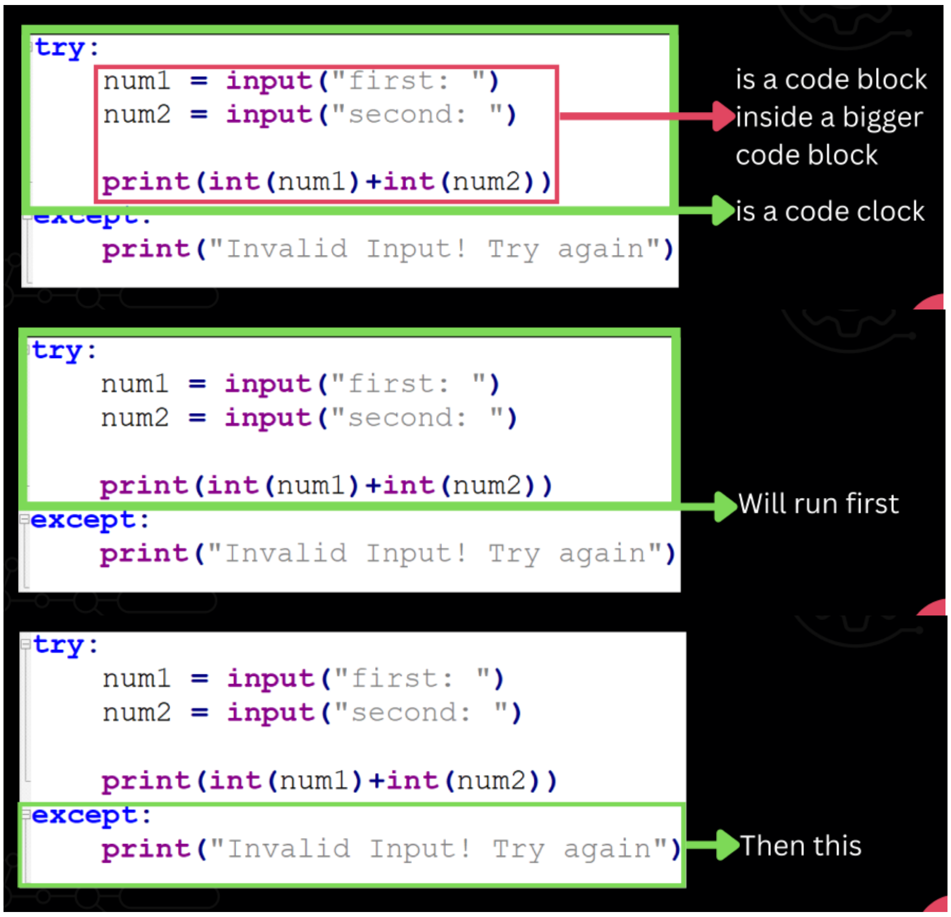
NULL

- This type is used to define a null variable or object.
- It makes use of the “None” keyword.
- We can assign “None” to any variable.
- It can also be used in Expressions.
    - emptyVal = None
    - print(emptyVal) # Outputs None

III. Error and Exception Handling

Code Block

- A block is a piece of Python program text that is run as a unit.
- A block is a smaller component of your program.
- Imagine it as beginning of a bigger command and the end.
- Code blocks are identified by their indentation in Python.



```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

is a code block inside a bigger code block

is a code clock

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Will run first

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

Then this

What can we conclude:

- In Python, the code is sequential
- It is modelled after a waterfall

- The code is read from top to bottom, from major code block to a minor code block one at a time.

What if we don't indent?

- Error

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

- Never use keywords or commands to name your variables.

EXCEPTION

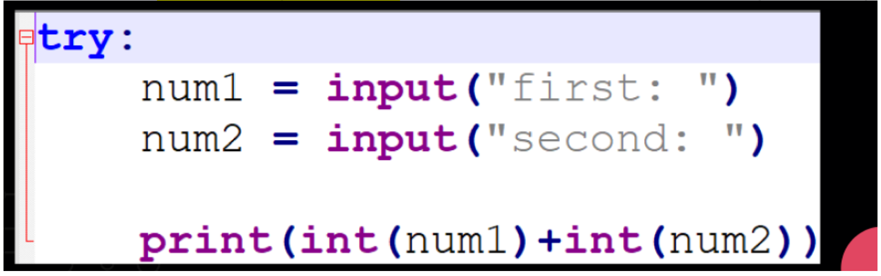
- It is an unwanted or unexpected event when a computer program runs.

EXCEPTION HANDLING

- It is the process of responding to unwanted or unexpected events when a computer program runs.
- When exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

TRY…

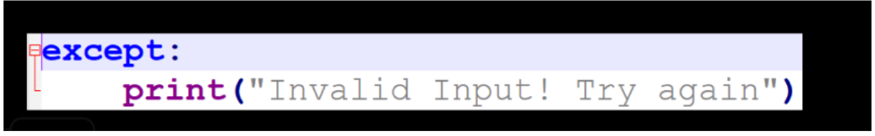
- In Python, exceptions can be handled using "try" statement.
- The critical operation which can raise an exception is placed inside the try clause.



```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
```

z

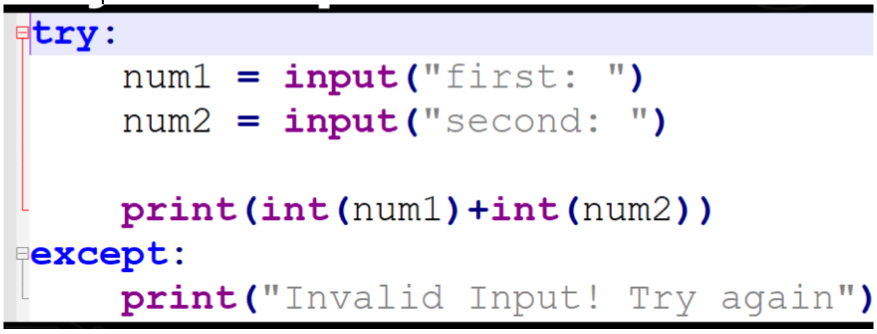
- The code that handles the exceptions is written in the except clause.
- We can thus choose what operations to perform once we have caught the exception.



```
except:
    print("Invalid Input! Try again")
```

TRY … EXCEPT

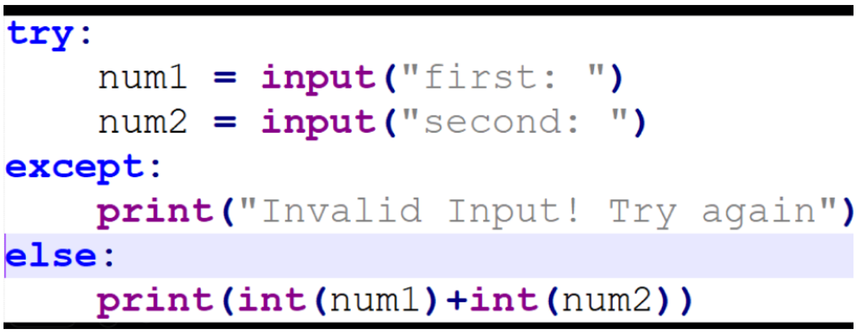
- Creates a block of code
- "Try" indents those proceeding after it until "Except" or another one of its optional keyword pair is used.



```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1)+int(num2))
except:
    print("Invalid Input! Try again")
```

TRY… ELSE

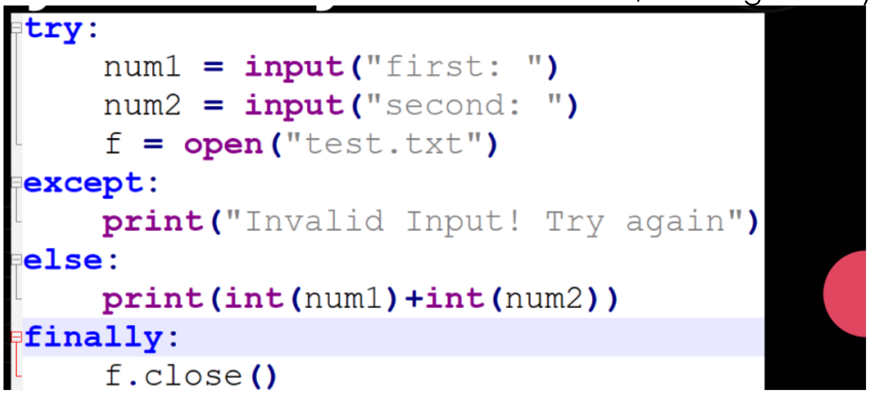
- You can use the optional "else" keyword with the try… except statement
- You may use this if you want code to run after the "try" statement did not have an error



```
try:
    num1 = input("first: ")
    num2 = input("second: ")
except:
    print("Invalid Input! Try again")
else:
    print(int(num1)+int(num2))
```

TRY … FINALLY

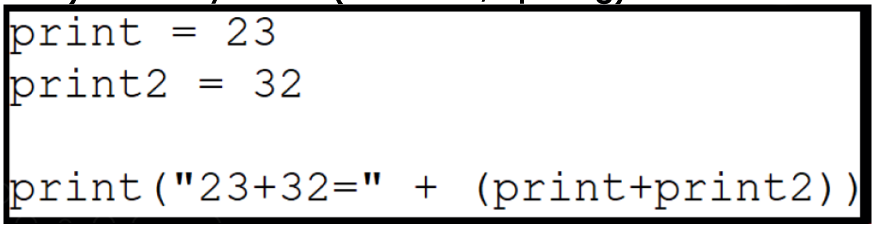
- The try statement in Python can have an optional finally clause.
- This block is executed no matter what, and is generally used to release external resources.



```
try:
    num1 = input("first: ")
    num2 = input("second: ")
    f = open("test.txt")
except:
    print("Invalid Input! Try again")
else:
    print(int(num1)+int(num2))
finally:
    f.close()
```

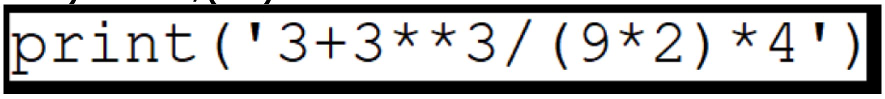
SYNTAX:
try:
        <content>
except:
        <what happens when exceptions occur>
<else:>
        <code to run if no exception in try>
<finally:>
        <code to run even if exception is done in try>

1) **Error – Syntactic (Grammar/ Spelling)**



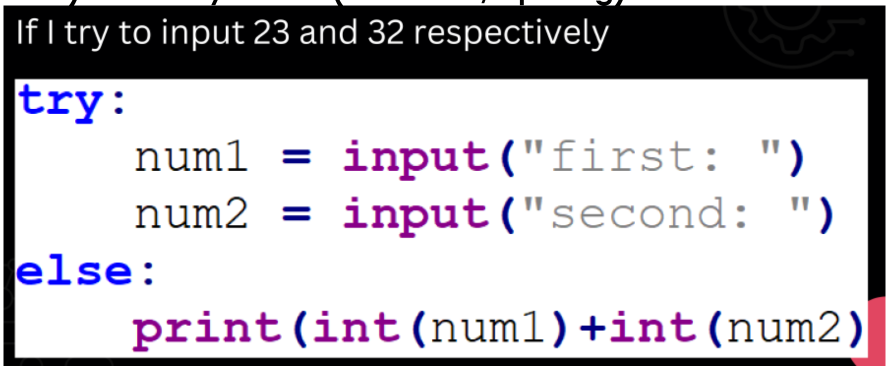
```
print = 23
print2 = 32

print("23+32=" + (print+print2))
```

2) 3+3**3/(8*2)*4



```
print('3+3**3/(9*2)*4')
```

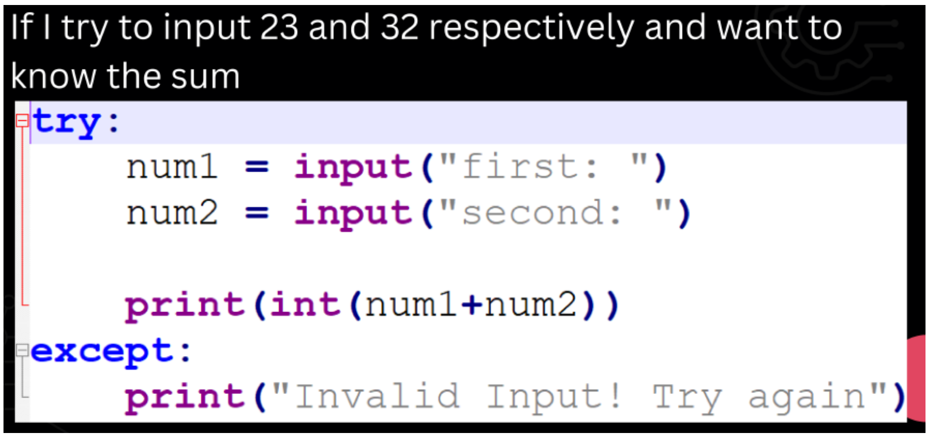
3) **Error – Syntactic (Grammar/ Spelling)**



If I try to input 23 and 32 respectively

```
try:
    num1 = input("first: ")
    num2 = input("second: ")
else:
    print(int(num1)+int(num2)
```

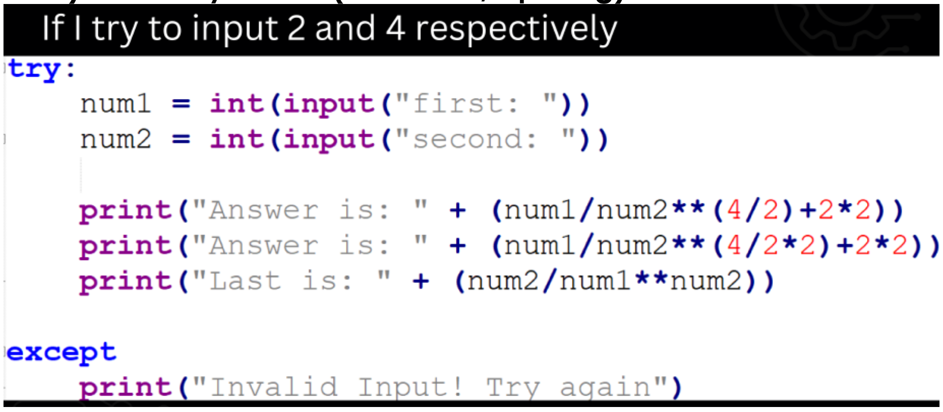
4) **Error – Semantic (Meaning/ Command)**



If I try to input 23 and 32 respectively and want to know the sum

```
try:
    num1 = input("first: ")
    num2 = input("second: ")

    print(int(num1+num2))
except:
    print("Invalid Input! Try again")
```

5) **Error – Syntactic (Grammar/ Spelling)**



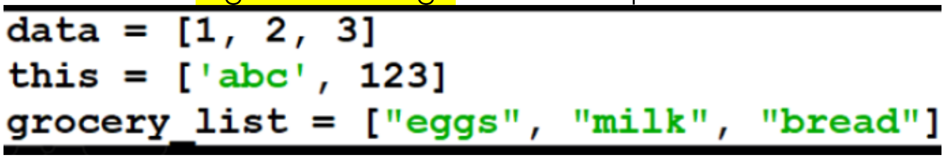
If I try to input 2 and 4 respectively

```
try:
    num1 = int(input("first: "))
    num2 = int(input("second: "))

    print("Answer is: " + (num1/num2**(4/2)+2*2))
    print("Answer is: " + (num1/num2**(4/2*2)+2*2))
    print("Last is: " + (num2/num1**num2))

except
    print("Invalid Input! Try again")
```

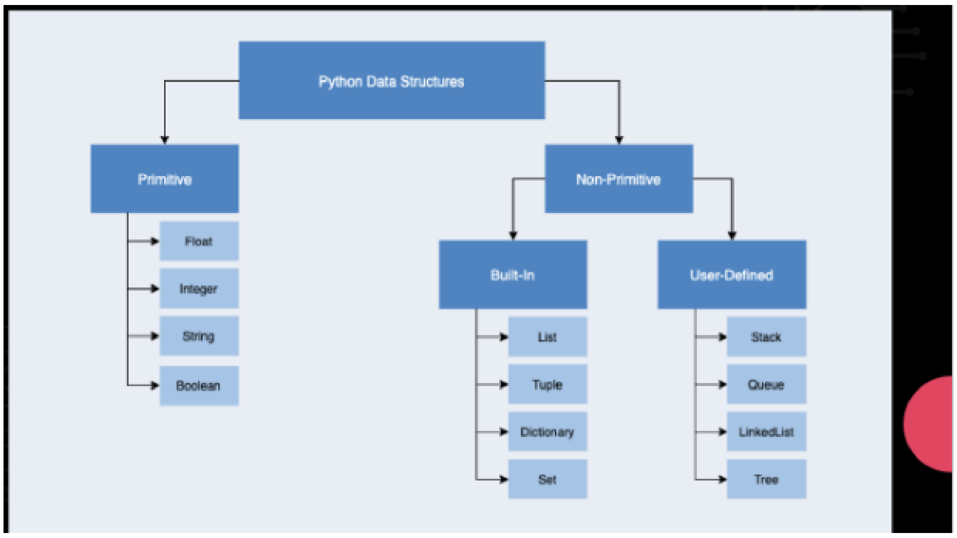
IV. CONDITION AND SEQUENCE

DATA STRUCTURES

- It is an organized storage used to keep data



```
data = [1, 2, 3]
this = ['abc', 123]
grocery_list = ["eggs", "milk", "bread"]
```



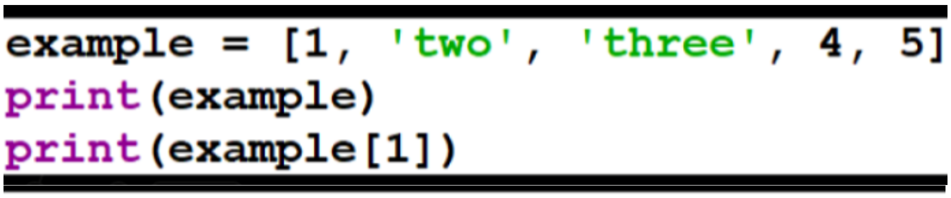
- An index refers to a position within an ordered list
- Think of it as some sort of location in a map
- For the built-in linear data structures. Think of it like how you number your quiz answers.

LISTS

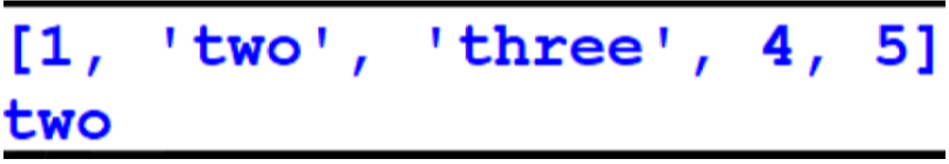
- It has an order (index 0 to n)
- May be a mixture of data types
- Items can be updated/overwritten

SYNTAX WHEN CREATING

- <list_name> = [<content1>, <content2>]



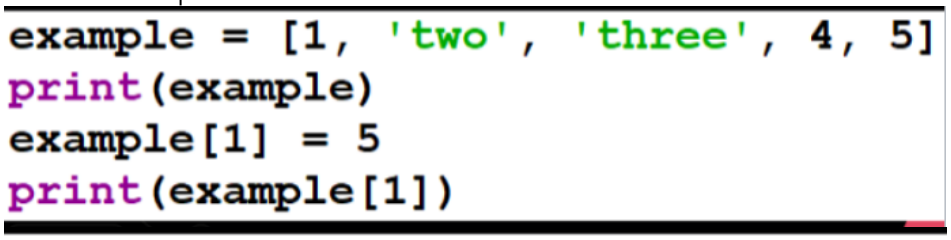
```
example = [1, 'two', 'three', 4, 5]
print(example)
print(example[1])
```



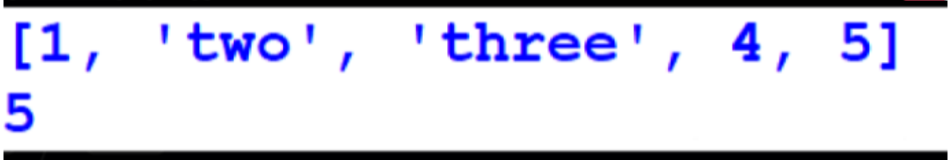
```
[1, 'two', 'three', 4, 5]
two
```

SYNTAX WHEN REFERENCING ITEM

- <list_name>[<index>]
- Example



```
example = [1, 'two', 'three', 4, 5]
print(example)
example[1] = 5
print(example[1])
```



```
[1, 'two', 'three', 4, 5]
5
```

CONCLUSION

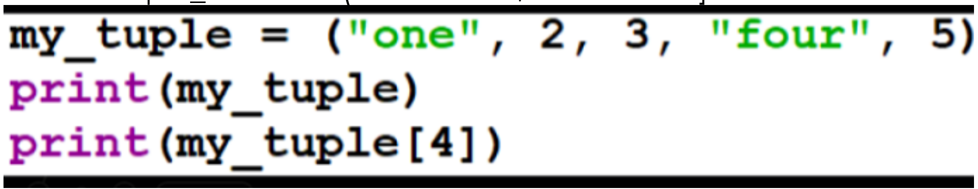
- Lists are created the same way as variables, just with squares brackets
- You can change the values of anything in the list
- We use indexes to refer to items in the list

TUPLE

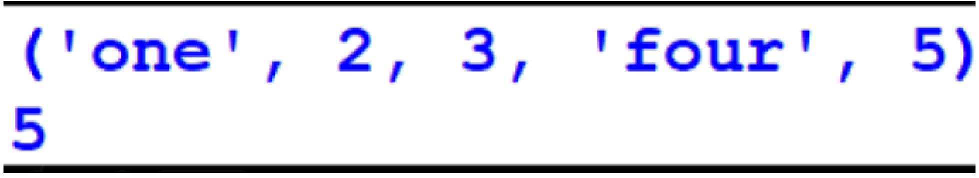
- It has an order (index 0 to n)
- May be a mixture of data types
- Items, once tuple is created, cannot be updated/overwritten

SYNTAX WHEN CREATING

- <tuple_name> = (<content1>, <content2>]



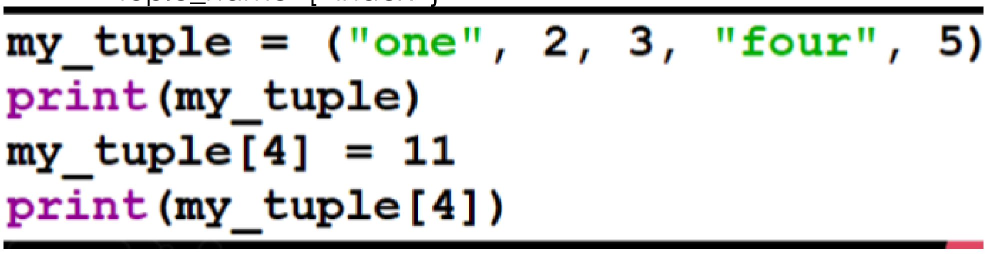
```
my_tuple = ("one", 2, 3, "four", 5)
print(my_tuple)
print(my_tuple[4])
```



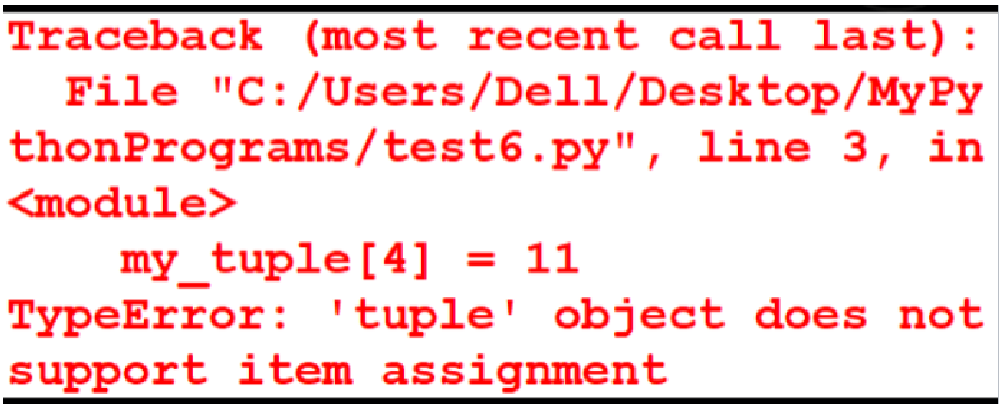
```
('one', 2, 3, 'four', 5)
5
```

SYNTAX WHEN REFERENCING ITEM

- <tuple_name>[<index>}



```
my_tuple = ("one", 2, 3, "four", 5)
print(my_tuple)
my_tuple[4] = 11
print(my_tuple[4])
```



```
Traceback (most recent call last):
  File "C:/Users/Dell/Desktop/MyPy
thonPrograms/test6.py", line 3, in
<module>
    my_tuple[4] = 11
TypeError: 'tuple' object does not
support item assignment
```

CONCLUSION

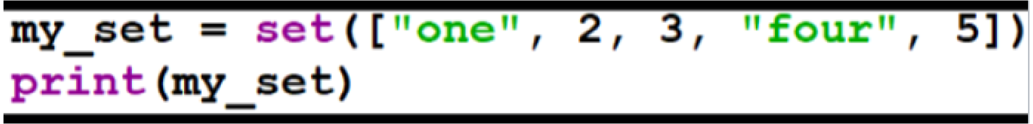
- Tuples are created the same way as variable, just with parentheses
- You cannot change the values of anything in the tuple
- We use indexes to refer to items in the tuple

SET

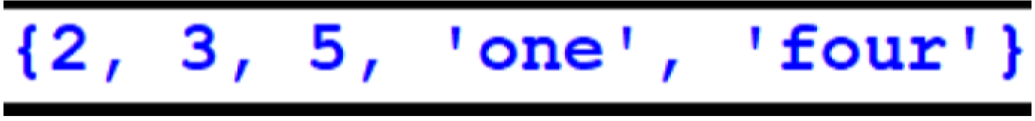
- It has no order
- May be a mixture of data types
- There can be no duplicate values, all items are unique
- Items can be overwritten/updated

SYNTAX WHEN CREATING

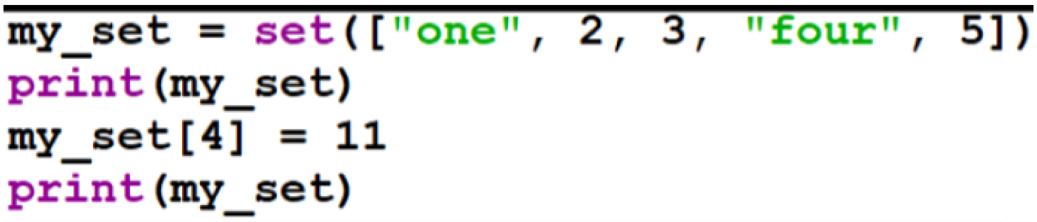
- <set_name> + set([<val1>, <val2>])
- Example



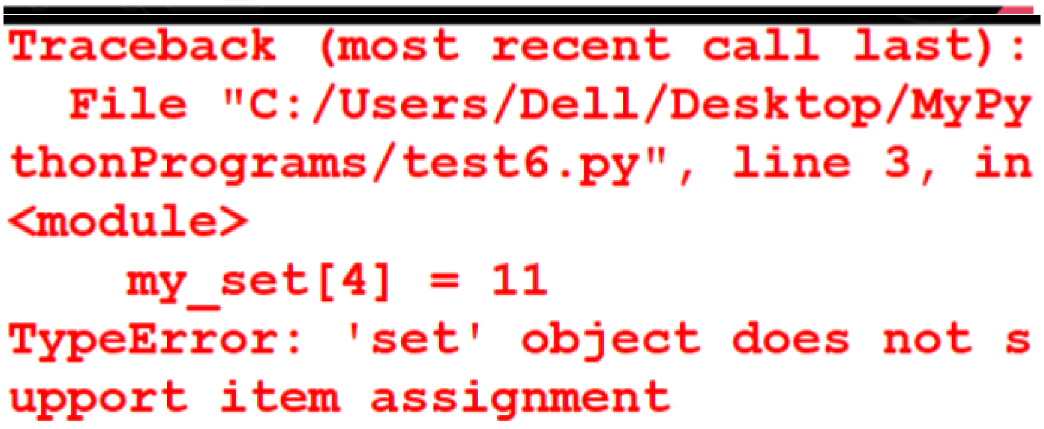
```
my_set = set(["one", 2, 3, "four", 5])
print(my_set)
```



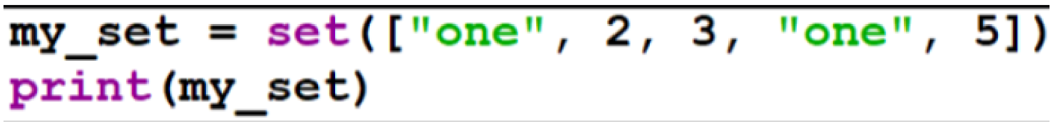
```
{2, 3, 5, 'one', 'four'}
```



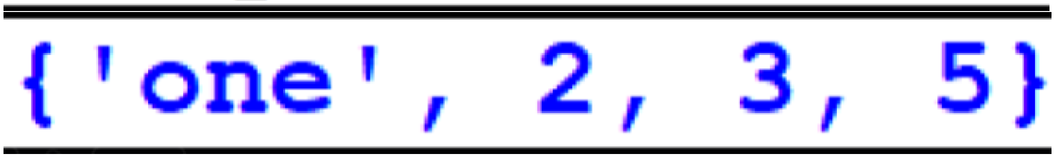
```
my_set = set(["one", 2, 3, "four", 5])
print(my_set)
my_set[4] = 11
print(my_set)
```



```
Traceback (most recent call last):
  File "C:/Users/Dell/Desktop/MyPy
thonPrograms/test6.py", line 3, in
<module>
    my_set[4] = 11
TypeError: 'set' object does not s
upport item assignment
```



```
my_set = set(["one", 2, 3, "one", 5])
print(my_set)
```



```
{'one', 2, 3, 5}
```

CONCLUSTION

- Presenting sets are done as a whole, they have no indexes assigned
- Sets sort themselves

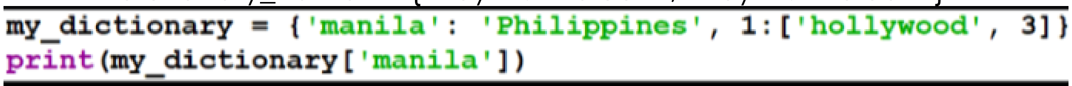
- Sets unify duplicate values

DICTIONARY

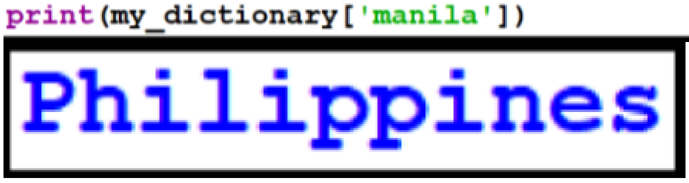
- It is ab unordered collection of data values (no indices)
- Dictionary holds the key: value pair
- Can use varied data types as key-value pairs

SYNTAX WHEN CREATING

- <dictionary_name> = {<key1>: <value1>, <key2>: <value2>}



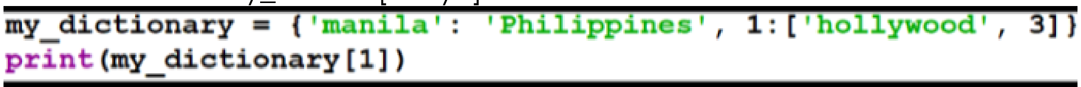
```
my_dictionary = {'manila': 'Philippines', 1:['hollywood', 3]}
print(my_dictionary['manila'])
```



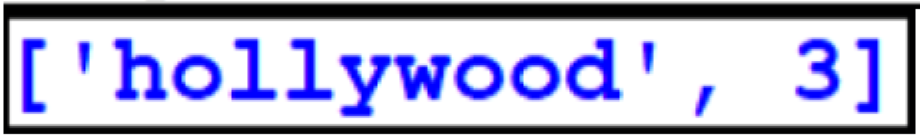
```
Philippines
```

SYNTAX WHENN REFERENCING ITEM

- <dictionary_name>[<key>]



```
my_dictionary = {'manila': 'Philippines', 1:['hollywood', 3]}
print(my_dictionary[1])
```



```
['hollywood', 3]
```

CONCLUSION

- Key-value pairs matter
- We may reference a key-value pair only by using the key
- Dictionaries use curly braces

REVIEW TABLE

- Fill In the following table in regards to the characteristics of the discussed Built-in Python Data Structures