# Python Workshop

Session 1 Outline

# Please Sign in at the Front

# Introduce orgs

ACM-W & Theta Tau


acm-w

# THETA TAU

## PROFESSIONAL ENGINEERING FRATERNITY

*Service – Profession – Brotherhood*

*Who Are We?*

Theta Tau is the oldest and largest professional engineering fraternity in the nation (est. 1904) and boasts an alumni network of 35,000+ initiated men and women

*Benefits of Joining:*

- Professional Development Workshops Hosted by Alumni
- Homework and Class Help
- Off Campus Housing

*For more information, contact Kenneth Gajefski at recruitment@ebthetatau.org*

# Installing Python

- Visual Studio Code (Not Visual Studio the IDE)
  https://code.visualstudio.com/Download
- Python 3 VS Code extension by Microsoft
- Python 3.8 and pip 3.8
  https://www.python.org/downloads/
- pylint
  ‘pip.exe install pylint’ within the powershell window

# Declaring Variables

Variables in python, unlike in C++, do not have to have their types stated when made. This is because C++ is a **strong and statically** typed language where python is **strong and dynamically** typed. Variables in python get their type depending on what you assign to them.

# Declaring Variables: Example

```
# Declare a variable and initialize it

#Integer
f = 99
print(f)

#String
g = 'Hello!'
print(g)

#Concatenate
a="AK"
b = 47
print (a+b) #Error! Cannot add two different types.

a="AK"
b = 47
print (a+str(b)) #Correct
```

# Strings

Just like in C++, strings in Python are immutable meaning that they **cannot** be modified in place. However, there are many modifying methods including lower(), upper(), join(), split(), find(), replace() etc., but strings can never be modified in place.

# String Methods: Example

```
1.  >>> "PrOgRaMiZ".lower()
2.  'programiz'
3.  >>> "PrOgRaMiZ".upper()
4.  'PROGRAMIZ'
5.  >>> "This will split all words into a list".split()
6.  ['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']
7.  >>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])
8.  'This will join all words into a string'
9.  >>> 'Happy New Year'.find('ew')
10. 7
11. >>> 'Happy New Year'.replace('Happy','Brilliant')
12. 'Brilliant New Year'
```

# Strings continued

```
var = “test123”
var.upper()
print(var)
# prints test123

var = “test123”
var = var.upper()
print(var)
# prints TEST123
```

In the first example var.upper() returns the value but is not re-assigned to the variable.

# Taking Inputs

The function `input()` displays a displays a desired string to the user and asks for an input from the user.

# Taking Inputs Example

**Example**:

variable = input(“What is your name?”)

Now whatever the user types after the prompt will be stored in the

“variable” variable.

# If - else

A **branch** is a program path taken only if an expression's value is true.

If branch: A branch taken only if an expression is true.

An if-else structure has two branches: The first branch is taken if an expression is true, else the other branch is taken.

# While loop

A **while loop** is a construct that repeatedly executes an indented block of code as long as the loop's expression is True.

At the end of the loop body, execution goes back to the while loop statement and the loop expression is evaluated again. If the loop expression is True, the loop body is executed again. But, if the expression evaluates to False, then execution instead proceeds to below the loop body.

Each execution of the loop body is called an **iteration**, and looping is also called iterating.

# For loop

A **for loop** statement loops over each element in a container one at a time, assigning the next element to a variable that can then be used in the loop body. The container in the for loop statement is typically a list, tuple, or string. Each iteration of the loop assigns the next element in the container to the name given in the for loop statement.

```
for variable in container:

    # Loop body: Sub-statements to execute

    # for each item in the container

# Statements to execute after the for loop is complete
```

# Functions

The way C++ determines if a line of code belongs under a loop or a function is to put it within the curly brackets for that function. The way python determines if a line of code belongs to a function or loop is through indents.

# Function Examples

**Example:**

In Python a function is defined using the def keyword:

```
Def my_function():
    print("Hello from a function")
```

To call a function, use the function name followed by parenthesis:

```
my_function()
```

# Main Function

The main function acts as the point of execution for any program. In Python, main function first **defines the main function**, then calls it in a **conditional statement**. The conditional uses a built in variable **“__name__”**, and checks if the function name is main.

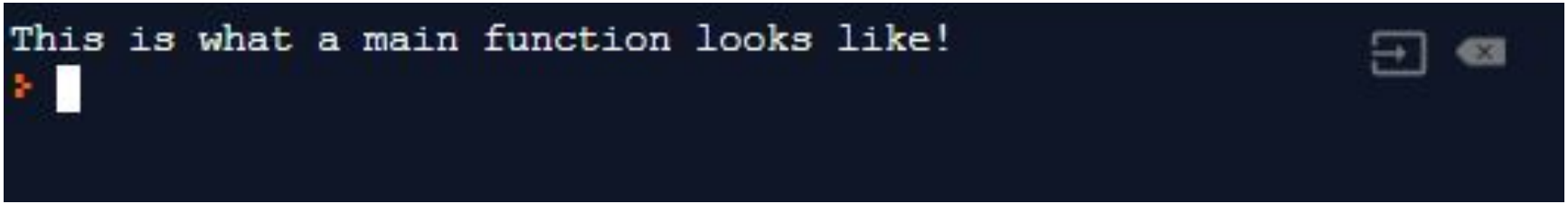
# Main Function: Example

```
# Declare and call the main function

#Main Function Definition
def main():
      print(“This is what a main function looks like!”)

If __name__ == “__main__”:
      main()

#Output:
```



# Lists, Dictionaries, and Tuples

**Lists** are a numbered list of values, with their numerical value starting from 0. You **can remove values** from the list and **add new values** to the end of it.

```
#List Ex: A list of the breakfast items in your kitchen
breakfast = [“Cereal”, “Milk”, “Eggs”, “Waffles”, “Coffee”]
print(breakfast)
```

# Tuples

**Tuples** are similar to lists, but their **values cannot be changed**. Like lists, the values are numbered starting from 0.

```
#Tuples Ex: Months in the year
months = (“jan”, “feb”, “mar”, “apr”, “may”, “jun”, “jul, “aug”, “sep”, “oct”, “nov”, “dec”)
```

# Dictionaries

**Dictionaries** are an index of elements, each of which has a definition. Each element is called a **key**, and each definition is called a **value**. Unlike lists and tuples, the keys are **not numbered**. They can be modified at any time.

```
#Dictionaries Ex: Save phone numbers
Contacts = {“Nidhi”:5551234, “Saloni”:2486957, “Hafsa”:7348325}
contacts[‘Tia’] = 3132001 #Adds another key
```

# Classes

**Classes** are user-defined prototypes to create objects. **Objects** are instance of classes, having all the properties defined in that class.

- Default functions and variables - public
- Add ‘_’ before name to make protected - Ex _num = 4
- Add ‘__’ before name to make private - EX __num = 4

```
class ClassName:
    # Statement-1
    # Statement-2
    # ...
    # Statement-N
```

# Classes

```
class Time:
    """ A class that represents a time of day """
    def __init__(self):
        self.hours = 0
        self.minutes = 0


time1 = Time()  # Create an instance of the Time class called time1
time1.hours = 7
time1.minutes = 30

time2 = Time()  # Create a second instance called time2
time2.hours = 12
time2.minutes = 45

print('{} hours and {} minutes'.format(time1.hours, time1.minutes))
print('{} hours and {} minutes'.format(time2.hours, time2.minutes))
```