



ellipsis



## Coders Assembly Workshop

18 Sep 2022

# Agenda

- Fundamentals
- Operator
- Strings
- Decision Logic
- Functions
- Exercise 1
- Looping
- List and Tuples
- Exercise 2



# Variable

- Container for data in program
- Can contain various types of data. The most basic type are:
  - Scalar -> Integer, Floating Point, Boolean, None
  - Non-scalar -> String
- In most programming language, including Python, `=` is used for assignment

```
# None Type Variable  
sample_variable = None
```

```
# String Variable  
name = 'Lance'  
name = "Lance's"
```

```
# Integer Variable  
enrolled_year = 2019
```

```
# Floating Point Variable  
gpa = 2.0
```

```
# Boolean Variable  
in_dean_list = False
```



# Variable

- Python is quite picky on naming
  - Must start with an alphabet (a-z, A-Z) or underscore ( \_ )
  - After that can contain any alphanumeric (a-z, A-Z, 0 - 9) or underscore ( \_ )
  - Cannot contain space in between character
  - Must not be a Python keyword  
([https://www.w3schools.com/python/python\\_ref\\_keywords.asp](https://www.w3schools.com/python/python_ref_keywords.asp))

```
this_is_normal_naming
```

```
# Examples of keywords
```

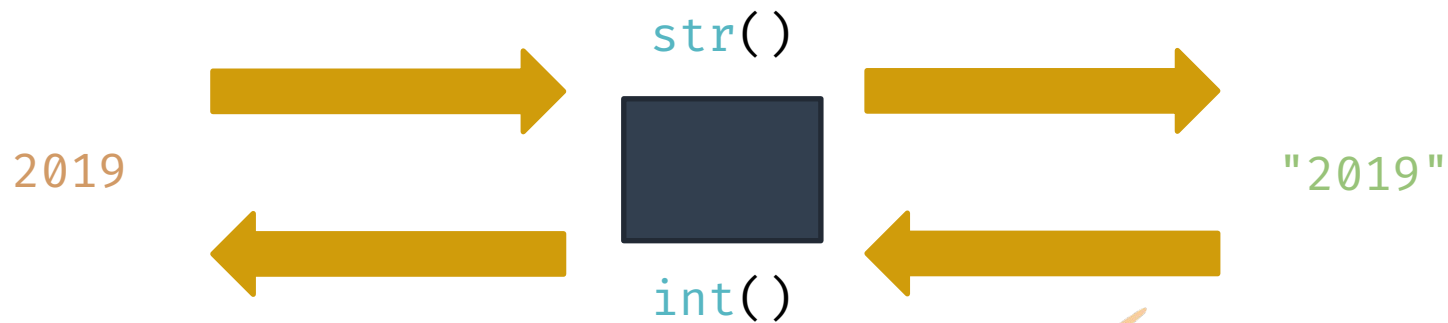
|         |      |          |
|---------|------|----------|
| False   | True | None     |
| finally | del  | continue |
| if      | from | while    |
| else    | elif | not      |



# Variable

- Depending on the content in the variable, you can convert their data type by using what we call typecasting. Here is a example of string to integer

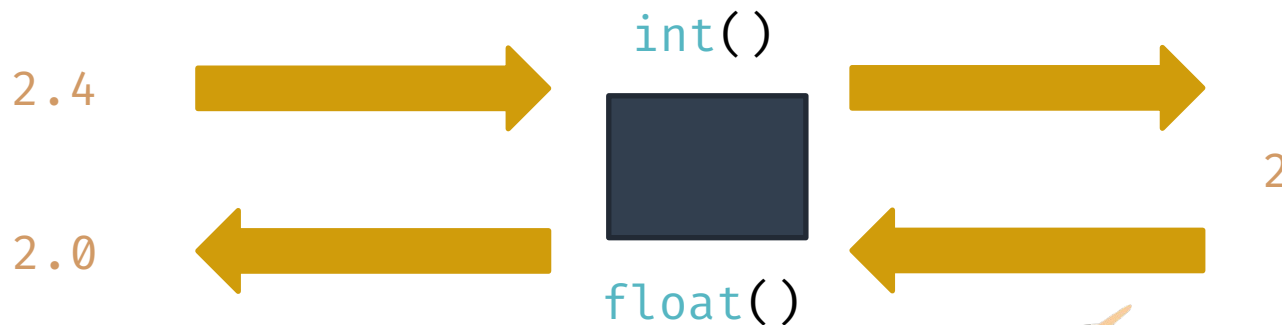
```
current_year = 2019  
result_str = str(current_year)  
result_int = int(result_str)
```



# Variable

- Similarly, a floating-point number can also be converted into a integer and vice versa. Do note that when you convert float to integer, you will lose the decimal information.

```
distance_km = 2.4  
distance_int = int(distance_km)  
distance_float = float(distance_int)
```



# Operators

- The Math Operators that you know about
- But symbol might be a little different in Python
  - Addition ( `+` )
  - Subtraction ( `-` )
  - Multiplication ( `*` )
  - Exponential ( `**` )
  - Floating Point Division ( `/` )
  - Integer Division ( `//` )
  - Modulo ( `%` )
- For floating point division, the result is always float.
- For any other operation, if one operand is a float, result is always float



# Precedence & Associativity

- It follows the precedence in mathematics
  - Parenthesis
  - Exponential
  - Multiplication / Division
  - Addition / Subtraction
- The way that math evaluate equations applies to most cases, from left to right

```
x = 5 / 2 * 2  
# Output  $\Rightarrow$  5.0
```

- One exception, assignment does right to left

```
x = 1  
y = 2  
z = 3  
x = y = z  
# x  $\Rightarrow$  3
```





# Strings Manipulation

- So what is the difference between single and double quote?

```
name = 'Lance'  
name = "Lance's"
```

- It is to allow the other quote to exist in the string, meaning
  - If you need single quote inside the string, use double quote to start and end
  - If you need double quote in the string, use single quote to start and end



# Strings Manipulation

- So, most of the time, your code editor will give you some hint, if there is an issue with your declaration.

```
string1 = 'CA's Workshop 1'  
string2 = "I say "Hello World""
```

- Note that there is a colour difference between the words. Those in green are considered captured as String but those in white isn't
- So how should we go about it



# Strings Manipulation

- Like we mentioned earlier, if you need to use single quote in the string, declare your string with a double quote, and vice versa

```
print ("CA's Workshop 1")  
print ('I say "Hello World"')
```

```
# Output  
CA's Workshop 1  
I say "Hello World"
```



# Strings Manipulation

- Alternatively, if you need both you can use this method call escaping string

```
print ('Cody\'s the instructor and he say "Hello World"')
```

```
# Output
```

```
Cody's the instructor and he say "Hello World"
```

- If you look carefully there's 2 character in the String that is in blue, this denotes that the character is “escaped” and is allowed to be printed
- Next, we will talk more about this escaping of Strings



# Strings Manipulation

- Here are a list of 'escapable' character that you should take note of

- `\n` ← Next Line
- `\t` ← Tab
- `\'` ← Print a single quote
- `\"` ← Print a double quote
- `\\` ← Print a slash

- We will be going thru everyone of them here in the next few slides



# Strings Manipulation

- `\n` ← Next Line

```
print ('line 1')  
print ('line 2')  
print ('-' * 10)  
print ('line 1\nline 2')
```

# Output

line 1

line 2

---

line 1

line 2

Ignore this line first, we  
will talk about it in later  
slides



# Strings Manipulation

- `\t` ← Tab

```
print ('1    2')  
print ('10   20')  
print ('-' * 10)  
print ('1\t2')  
print ('10\t20')
```

# Output

```
1    2  
10   20  
-----  
1    2  
10   20
```



# Strings Manipulation

- `\'` ← Print a single quote

```
print ('O\'Reilly')
```

```
# Output  
O'Reilly
```





# Strings Manipulation

- `\"` ← Print a double quote

```
print ("\\"Computers are like Old Testament gods: lots of rules and no mercy.\" - Joseph Campbell")
```

# Output

```
"Computers are like Old Testament gods: lots of rules and no mercy." - Joseph Campbell
```



# Strings Manipulation

- `\\` ← Print a slash

```
print ('d\d')  
print ('d\\d')  
print ('n\\n')
```

# Output

```
d\d  
d\d  
n\n
```



# Strings Formatting

- To print out a string, there are a few ways to combine variables into the string.
- First method is the old way of format string. To use it, after a string, you will put `.format()` to denote

```
name = 'Lance'  
age = 24  
print ( 'My name is {} and my age {}'.format(name, age) )
```



# Strings Formatting

- To print out a string, there is a few ways to combine variables into the string.
- First method is the old way of format string. To use it, after a string, you will put `.format()` to denote

```
name = 'Lance'  
age = 24  
print ( 'My name is {} and my age {}'.format(name, age) )
```

- Within the string, for every variable that you need, you will put a `{}` and within the parenthesis of the `format` method



# Strings Formatting

- The other method is using the what we call f-string. To use it, you would need to put the letter f in front of the string.

```
name = 'Lance'  
age = 24  
print ( f'My name is { name } and my age { age }' )
```

- To place the variable, you will place the variable name within the `{ }`



# Strings Operations

- Earlier on, I wrote this in the code `print ('-' * 10)`
- This is known as String Operation. There's 2 main operation in String
  - `+` to do String Concatenation
  - `*` to do String Repetition

```
string_1 = 'Hello'  
string_2 = 'World'  
print (string_1 + string_2)  
print (string_1 * 3)
```

```
# Output  
HelloWorld  
HelloHelloHello
```



# Strings Slicing

- At the very start, we also talk about String being a Non-Scalar type. So what does it mean?
  - Non-scalar means it is a data structure with multiple data points
  - A String can have many character, so to know how long is the string you can use this function `len()`

```
string_1 = 'Hello World'  
print ( len(string_1) )
```

```
# Output  
11
```



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|----|
| H | e | l | l | o |   | W | o | r | l | d  |

# Strings Slicing

- So how does knowing the length help?
  - Once you know the length, you can find the position which is called index
  - By knowing the index, we can do something call String Slicing. It is a process of extracting part of the string out from a bigger string.
  - In programming, counting starts from 0. So, in our previous example, each character is known to be stored at an index from 0 to 10

```
string_1 = 'Hello World'
print ( string_1[0] )
print ( string_1[1] )
print ( string_1[6] )
print ( string_1[7] )
```

# Output

H  
e  
W  
o





# Strings Slicing

- String Slicing not only can extract a character but also part of a string.
- To do slicing, it is fairly similar to how we pull out a character, but with different parameter
- `string_1[start:stop:step]`
  - Start -> Refers to which index to start from (Index inclusive)
  - Stop -> Refers to which index to stop at (Index exclusive)
  - Step -> What is the interval of extraction
- The next few slides will talk more about the various parameter



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|----|
| H | e | l | l | o |   | W | o | r | l | d  |

## Strings Slicing – Start

- Start -> Refers to which index to start from (Index inclusive)

```
string_1 = 'Hello World'  
print(string_1[6:])
```

```
# Output  
World
```

- What the above code states is that we will start slicing from Index 6. You might ask why is there no Stop value since the `:` is meant to separate the start and stop value.
- By default, if stop value is omitted, it will slice until the end of the string



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|----|
| H | e | l | l | o |   | W | o | r | l | d  |

## Strings Slicing – Stop

- Stop -> Refers to which index to stop at (Index exclusive)

```
string_1 = 'Hello World'  
print (string_1[:4])
```

```
# Output  
Hell
```

- What the above code states is that we will stop slicing at Index 4. Similarly, when start value is omitted, the string will start slicing from the very front of the string
- The stop is at 4 but why is the output **Hell** instead of **Hello**
- Do always take note, Stop value is always exclusive, meaning the character at that index will NOT be extracted



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|----|
| H | e | l | l | o |   | W | o | r | l | d  |

## Strings Slicing – Step

- Step -> What is the interval of extraction

```
string_1 = 'Hello World'  
print(string_1[::2])
```

```
# Output  
HloWrđ
```

- Now, in the earlier slides, we did not indicate step. This is because by omitting step, it is defaulted to **1** which means every character
- When we say step of **2** it means every other character
- One thing to take note is that, here we omitted the start and stop. As mentioned in the previous slides, if these are omitted
  - Start will be the very front, aka Index 0
  - Stop will be the very end, aka last Index value



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|----|
| H | e | l | l | o |   | W | o | r | l | d  |

## Strings Slicing – Step

- Step -> What is the interval of extraction

```
string_1 = 'Hello World'
print(string_1[::-1])
```

```
# Output
dlroW olleH
```

- If lets say you need to flip the value around, what you can do is indicate a **-1** in the step and it will flip it the other way around.



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|----|
| H | e | l | l | o |   | W | o | r | l | d  |

# Strings Slicing

- Some point to note
  - When you only indicate the index to extract out a character, you cannot indicate a index that is non-existent
  - But you can slice a non-existent index
  - You can also stop at a non-existent index, it will give you until the last character of the string

```
string_1 = 'Hello World'
print (string_1[12]) # Error
```

```
print (string_1[12:])
# Output
```

← This is a empty string

```
print (string_1[6:12])
# Output
World
```



|     |     |    |    |    |    |    |    |    |    |    |
|-----|-----|----|----|----|----|----|----|----|----|----|
| -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |
| H   | e   | l  | l  | o  |    | W  | o  | r  | l  | d  |

# Strings Slicing – Negative Indexing

- So far for start and stop we have been counting from left to right
  - This is known as positive indexing
- We will now talk about negative indexing, which is counting from right to left
- But take note
  - Start < Stop value, if not you will get an empty string
  - Stop value is still exclusive

```
string_1 = 'Hello World'
print (string_1[-11:-7])
print (string_1[-5:])
```

```
# Output
Hell
World
```



# Decision Logic

- Boolean data type is the most important part that you need to understand when doing decision logic
- Boolean contains either `True` or `False`
- All comparison will result in a Boolean. Comparison that you can make includes:

```
<    # less than
≤    # less than or equals
=    # equals to
≠    # not equals to
≥    # more than or equals
>    # more than
```





# Decision Logic

- Here are some example of the output

```
# Python Codes  
print( 3 == 2 )  
print( 3 >= 2 )  
print( 3 != 2 )
```

```
# Output  
False  
True  
True
```

```
# Python Codes  
num1 = 3  
num2 = 2
```

```
print( num1 == num2 )  
print( num1 >= num2 )  
print( num1 != num2 )
```

```
# Output  
False  
True  
True
```



# Decision Logic

- In programming, we can indicate certain condition before we do certain stuff. In Python, there is 3 keywords that you need to take note to indicate such logic

- `if` conditions:
- `elif` conditions:
- `else`:

- Often mistakes are made for decision logic due to carelessness. For the following code, what is the mistake made, and the output when corrected?

```
# Following line has a error  
age = input('What's your age')  
print (age <= 16) # True or False
```



# Decision Logic

- So now that we have corrected the code, this is how you would usually use this kind of condition.

```
# Correct way of putting it
age = int( input("What's your age") )
age = int( input('What\'s your age') )

if age ≤ 16:
    print ("You're still underage")
elif age ≥ 55:
    print ("Can take CPF already")
else:
    print ("The awkward age")
```



# Decision Logic

- Just need to be cautious that, if else statements runs one by one. So, the following code might not behave as you expected

```
# Correct way of putting it
age = int( input("What's your age") )
age = int( input('What\'s your age') )

if age ≤ 16:
    print ("You're still underage")
elif age ≥ 21:
    print ("Legal liao wor")
elif age ≥ 55:
    print ("Can take CPF already")
else:
    print ("The awkward age")

# In this situation, even if you input 55 or more, it will print "Legal liao wor" instead of "Can take CPF already".
```



# Functions

- Functions are used to consolidate repetitive blocks of codes which serve a certain purpose.

```
string_1 = 'Hello World'
```

```
print (string_1)  
print ('-' * 10)  
print (string_1)  
print ('-' * 10)  
print (string_1)
```

```
# Output  
Hello World  
_____  
Hello World  
_____  
Hello World
```

For someone that read your code, they might not know what is the purpose of this.



# Functions

- Functions are used to consolidate repetitive blocks of codes which serve a certain purpose.

```
string_1 = 'Hello World'

print (string_1)
print ('-' * 10)
print (string_1)
print ('-' * 10)
```

```
# Output
Hello World
_____
Hello World
_____
```

```
def print_helloWorld_seperator():
    string_1 = 'Hello World'
    print (string_1)
    print ('-' * 10)

print_helloWorld_seperator()
print_helloWorld_seperator()
```

```
# Output
Hello World
_____
Hello World
_____
```



# Functions

- How do you construct a function
  - All functions starts with a keyword `def` followed by the name of the function.
  - In our earlier example `print_helloWorld_separator` is the name of the function
  - All function ends of with a `()`
- So why do the function ends with the `()` ?
  - Remember your  $f(x)$  in math? It's the same concept whereby between the parenthesis it can take in parameter.
  - So now we will see how to use this in the next few slides



# Functions

- We can see here the function `print_power_2` takes in a variable call `x` and prints out the value of `x` square
- In functions we can also make use of a keyword call `return` that will pass back the value to the line that calls it. But by using return, you need to store the value in a variable

```
def print_power_2 (x):  
    print (x * x)  
  
print_power_2(4)  
  
# Output  
16  
  
def print_power_2_return (x):  
    return x * x  
  
result = print_power_2_return(4)  
print (result)  
  
# Output  
16
```





# Exercise 1



# Looping

- In programs we often need to repeat certain processes or codes. However, it is a bad practice where we copy and paste codes. Let's say if I want to print a statement for 5 times and I do copy and paste
- What if now I want to print out 20 times?



# Looping

- So in Python, we use `for` loop to repeat certain codes

```
# Template Code
for i in range (start, stop, step):
    print ('Hello World')
```

- For any counting in programming, we start counting from 0
- Typically, for loop we will only indicate the stop value to denote how many times we want it to run

```
# Only include stop value
for i in range (5):
    print ('Hello World', i)
```

```
# Output
Hello World 0
Hello World 1
Hello World 2
Hello World 3
Hello World 4
```



# Looping

- But what can do is also to indicate the start and stop values when you need to use the number that is stored in the counter ( In our example it would be the variable `i` )

```
# Only include start and stop  
for i in range (1, 5):  
    print ('Hello World', i)
```

```
# Output  
Hello World 1  
Hello World 2  
Hello World 3  
Hello World 4
```

- By default, if you did not indicate the step, it will always be increment by 1



# Looping

- What if you want to only do every even number from 2 to 10? Or you want to count backwards? You will need to indicate the steps as the third parameter.

```
# Include all parameters
# (Counting backwards)
for i in range(10, 0, -1):
    print('Hello World', i)
```

```
# Output
Hello World 10
Hello World 9
Hello World 8
Hello World 7
Hello World 6
Hello World 5
Hello World 4
Hello World 3
Hello World 2
Hello World 1
```

```
# Include all parameters
# (Skipping count)
for i in range(2, 11, 2):
    print('Hello World', i)
```

```
# Output
Hello World 2
Hello World 4
Hello World 6
Hello World 8
Hello World 10
```



# Looping

- You must be wondering why these two examples look so weird.
  - Example 1 I want to stop at 10 but code says stop at 11
  - Example 2 stop at 5 but print until 4
- In Python, start value is **inclusive** but stop value is **exclusive**

```
# Eg 1
for i in range (2, 11, 2):
    print ('Hello World', i)
```

```
# Output
Hello World 2
Hello World 4
Hello World 6
Hello World 8
Hello World 10
```

```
# Eg 2
for i in range (1, 5):
    print ('Hello World', i)
```

```
# Output
Hello World 1
Hello World 2
Hello World 3
Hello World 4
```



# Looping

- For loop can also be used in Non-Scalar data type like String or List

```
# Template Code
for ch in 'Hello World':
    print (ch)
```

# Output

H  
e  
l  
l  
o  
  
W  
o  
r  
l  
d

```
# Template Code
for num in [1,2,3]:
    print (num)
```

# Output

1  
2  
3



# Nested Looping

- In all loops, you are able to run more loops within it, this is what we call nested looping.
- How you can do it is just simply declaring more for loops within the for loops if you need them

```
for i in range(5):  
    result = str(i+1) + ' | '  
    for j in range(5):  
        result += str( (i+1) * (j+1) ) + ' '  
    print (result)
```

# Output

|   |  |   |    |    |    |    |
|---|--|---|----|----|----|----|
| 1 |  | 1 | 2  | 3  | 4  | 5  |
| 2 |  | 2 | 4  | 6  | 8  | 10 |
| 3 |  | 3 | 6  | 9  | 12 | 15 |
| 4 |  | 4 | 8  | 12 | 16 | 20 |
| 5 |  | 5 | 10 | 15 | 20 | 25 |





# Nested Looping

- In all loops, you are able to run more loops within it, this is what we call nested looping.
- How you can do it is just simply declaring more for loops within the for loops if you need them

```
for i in range(5):  
    result = ''  
    for j in range(i+1):  
        result += str( j+1 )  
    print (result)
```

# Output

```
1  
12  
123  
1234  
12345
```



# Nested Looping

- Let's breakdown the code
  - `for i in range(5):`
    - This is to run the for loop for 5 times with `i` containing 0, 1, 2, 3, 4 at respective run
  - So when `i` is 0, what does `for j in range(i+1):` means?
    - This means the for loop will run for 1 time with `j` containing 0 when `i` is 0

```
for i in range(5):  
    result = ''  
    for j in range(i+1):  
        result += str( j+1 )  
    print (result)
```



# Nested Looping

- The following will be the subsequent behaviour for `for j in range(i+1):`
  - When `i` is 1, `j` runs 2 times containing 0, 1 at respective run
  - When `i` is 2, `j` runs 3 times containing 0, 1, 2 at respective run
  - When `i` is 3, `j` runs 4 times containing 0, 1, 2, 3 at respective run
  - When `i` is 4, `j` runs 5 times containing 0, 1, 2, 3, 4 at respective run
- Note that when `i` changes value, the variable `result` is also reset back into a empty string

```
for i in range(5):  
    result = ''  
    for j in range(i+1):  
        result += str( j+1 )  
    print (result)
```



# List & Tuples

- List & Tuples are a basket of values. They are usually of basic data type, or a nested list (not really tested)

```
list_1 = ['Hello', 'World']
```

```
print (list_1[0])  
print ('-' * 10)  
print (list_1[1])  
print ('-' * 10)
```

```
# Output
```

```
Hello
```

```
_____
```

```
World
```

```
_____
```



| 0     | 1     | 2        |
|-------|-------|----------|
| Hello | World | Ellipsis |

## List & Tuples

- They are access by their index like accessing character in String data type

```
list_1 = ['Hello', 'World', 'Ellipsis']
```

```
print (list_1[0])  
print ('-' * 10)  
print (list_1[1])  
print ('-' * 10)
```

```
# Output  
Hello
```

```
_____  
World
```



| 0     | 1     | 2        |
|-------|-------|----------|
| Hello | World | Ellipsis |

## List & Tuples

- Tuples are denoted using ( and ). The only difference is that they are immutable

```
list_1 = ('Hello', 'World', 'Ellipsis')
```

```
print (list_1[0])  
print ('-' * 10)  
list_1[0] = 'Bye'  
print (list_1[1])  
print ('-' * 10)
```

```
# Output  
Hello
```

```
TypeError: 'tuple' object does not support item assignment
```



**ellipsiS**  
SMU INFORMATION SYSTEMS SOCIETY

| 0     | 1     | 2        |
|-------|-------|----------|
| Hello | World | Ellipsis |

## List & Tuples

- Usually we will iterate through the list using a for loop

```
list_1 = ('Hello', 'World', 'Ellipsis')
```

```
for element in list_1:  
    print (element)
```

```
# Output
```

```
Hello
```

```
World
```

```
Ellipsis
```



**ellipsiis**  
SMU INFORMATION SYSTEMS SOCIETY

# Exercise 2





# Tips for Lab 1



# Thank You



**ellipsiis**  
SMU INFORMATION SYSTEMS SOCIETY