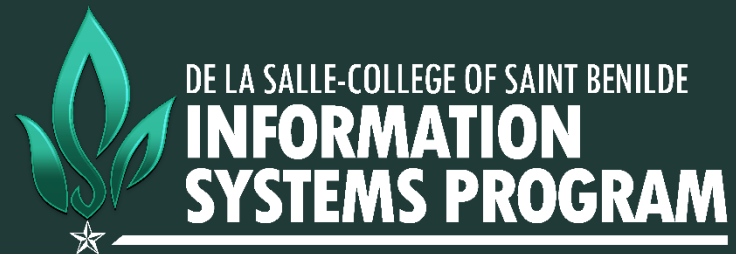


Application Development and Emerging Technologies (APPDAET)

Jino Raymundo



Data Privacy Notice

De La Salle-College of Saint Benilde (DLS-CSB) respects your right to privacy and is committed to protect the confidentiality of your personal information thus has adapted necessary organizational, technical and physical measures to secure it. DLS-CSB is bound to comply with the Data Privacy Act of 2012 (RA 10173), its implementing Rules and Regulations and relevant issuances of the National Privacy Commission.

By participating in this video meeting/conference/webinar, you are consenting to our collection(including recording) in accordance with this Privacy Notice. Information is also processed via this video conferencing platform. Please refer to the privacy policy found in their website.

The information processed: name, email address, your image, video and audio will be used for attendance, documentation, communication and systems administration purposes.

Only authorised individuals of Department/Office/Unit organizing the event and other offices authorized will have access to this information and will not be disclosed to third parties without your approval.

We also use information gathered from you for abuse prevention and privacy protection.

DLS-CSB shall only retain the said personal information until it serves its purpose, after which it shall be securely disposed of.

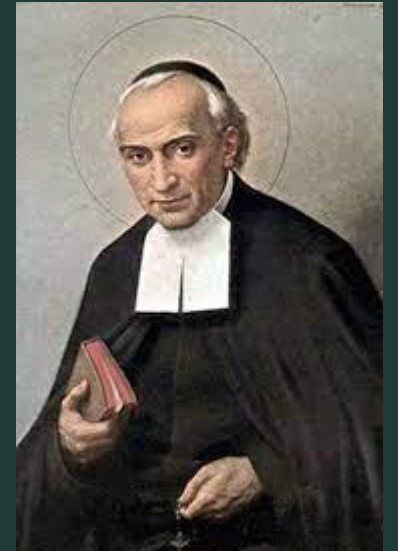
Queries and complaints can be directed to the Data Protection Officer at dpo@benilde.edu.ph.

Our Lasallian Prayer

I will continue, o my God, to do all my actions for the love of You.



Mary, our Lady of the Star, pray for us.
Saint John-Baptiste de La Salle, pray for us.
Saint Benilde Romançon, pray for us.



Live, Jesus, in our hearts, forever!

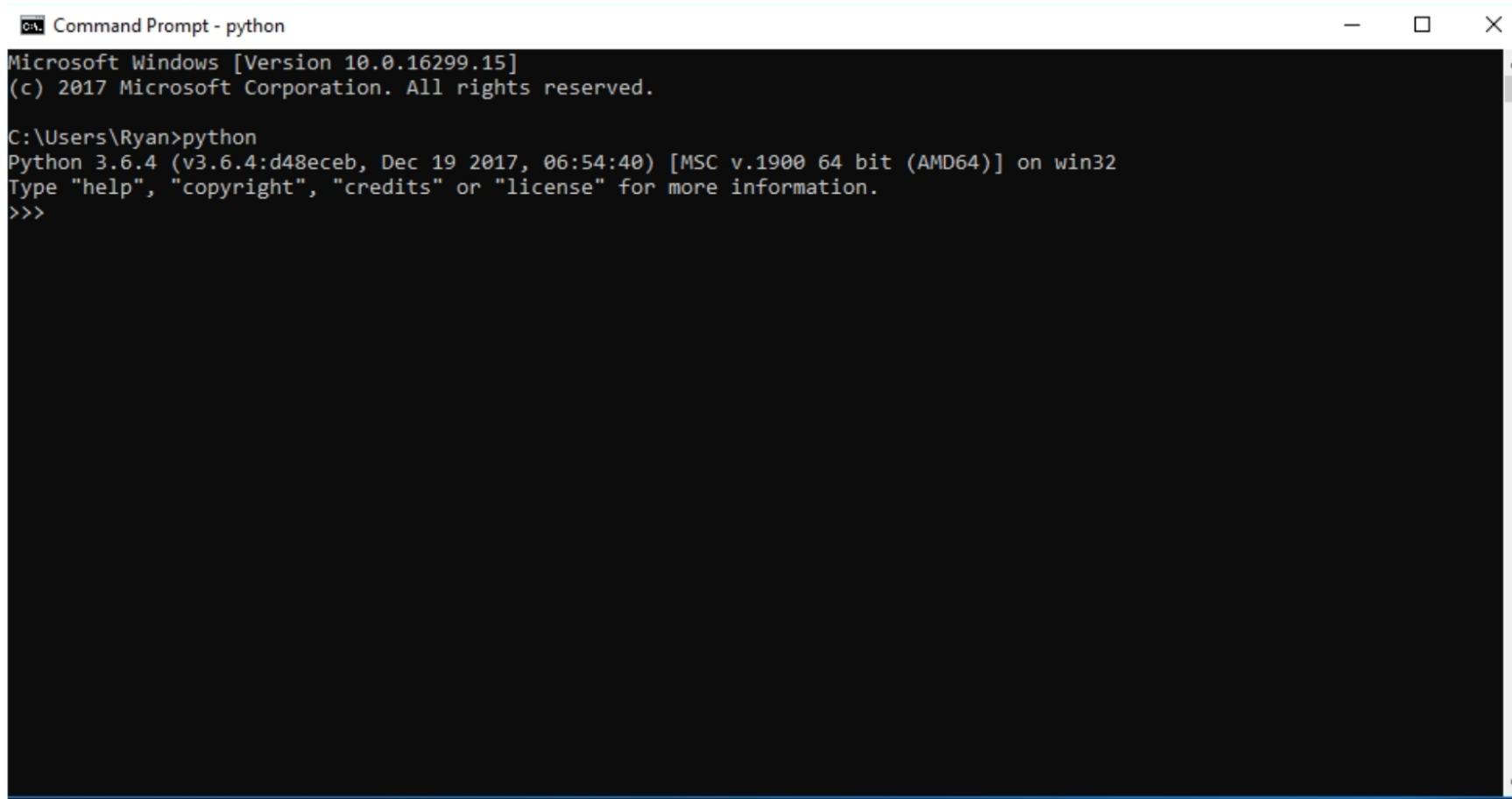
Lesson Outline

- Introduction to the Python Language
- Python Syntax
- Variables
- User Input, Comments and Indentations

What is Python?

- Python is a ***high-level, general-purpose*** programming language
 - Notorious for having a very simple “pseudocode-like” syntax that places emphasis on readability and expressiveness
- Development is faster because it is ***interpreted*** (as opposed to ***compiled***)
- Supports multiple paradigms:
 - Object-oriented programming
 - Functional programming
 - Imperative programming
 - Procedural programming

Checking your Python Installation



```
Command Prompt - python
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Ryan>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Hello World in Python

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

When using the Python Interpreter, the `>>>` is the “Python prompt” where you can send commands directly into the interpreter.

You can also write scripts by saving Python commands in a `.py` file.

Python Syntax

- Python has many similarities to Perl, C and Java, but there are some definite differences:
 - In Python, you can do “Interactive Mode” or “Script Mode”
 - Python Identifiers are similar:
 - Should start with a letter (A-Z or a-z) or an underscore (_)
 - Followed by zero or more letters, underscores and digits (0-9)
 - Python is **case-sensitive**
 - Cannot be a **Python reserved keyword**
 - Naming conventions are as follows:
 - Class names start with an uppercase letter. All other identifiers start with a lowercase letter
 - Starting an identifier with a **single leading underscore** (_) means that the identifier is **private**.
 - Starting an identifier with **two leading underscores** (__) indicates a **strong private** identifier
 - If the identifier also ends with **two trailing underscores**, the identifier is a **language-defined special name**.

Python Reserved Keywords

- In **interactive mode**, you can get the full list of keywords by entering **help()**, then entering **keywords**
- Exit help mode by entering **quit**

```
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help()
```

```
Welcome to Python 3.11's help utility!
```

```
If this is your first time using Python, you should definitely check out
the tutorial on the internet at https://docs.python.org/3.11/tutorial/.
```

```
Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules. To quit this help utility and
return to the interpreter, just type "quit".
```

```
To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics". Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
```

```
help> keywords
```

```
Here is a list of the Python keywords. Enter any keyword to get more help.
```

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

```
help> |
```

Variables

- Variable names can hold a ***value***
- Variables ***do not need to be declared in advance***
- Variables with compound names (i.e. a variable with two words) can be separated using underscore (_)

```
>>> first_letter = "a"
```

- Constants should be written in ***capital letters*** while also separating compound names using underscore (_)
- Assign values to variables using the **assignment operator** (=)

User Input and Output

- To accept input, assign a variable with the **input()** function:

```
>>> message = input()
```

- Display any output by using the **print()**

```
>>> print(message)  
Peter Piper picked a peck of pickled peppers  
>>>
```

Comments

- You can add comments in your Python scripts by beginning it with the **pound sign (#)**
 - A ***block comment*** comes in the line before the statement it annotates and is placed at the same indentation level:

```
# increment counter  
counter = counter + 1
```

- An ***inline comment*** is placed on the same line as the statement it annotates

```
>>> print(foobar) # this will raise an error since foobar isn't defined
```

Documentation Strings

- A document string (or ***docstring***) is a literal string used as a Python comment.
- It is written and wrapped within ***triple quotation marks*** (""")
- Often used to document modules, functions and class definitions, or to write ***multi-line comments***

```
"""
This script can be called with two integer arguments to return their sum
"""

import sys
num_1 = int(sys.argv[1])
num_2 = int(sys.argv[2])

print(num_1, "+", num_2, "=", num_1 + num_2)
```

Indentation

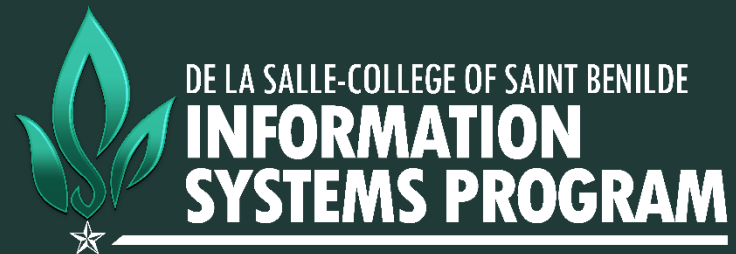
- A **block** is a group of statements that are meant to be executed together.
- Unlike in C#/Java which uses { }, blocks are defined in Python **based on the indentation**

```
if True:
    # execute this block of statements
    print("Block 1")
else:
    # execute other block of statements
    print("Block 2")
```

- Therefore, ***proper indentation is very important in Python!***

Application Development and Emerging Technologies (APPDAET)

Jino Raymundo



Data Privacy Notice

De La Salle-College of Saint Benilde (DLS-CSB) respects your right to privacy and is committed to protect the confidentiality of your personal information thus has adapted necessary organizational, technical and physical measures to secure it. DLS-CSB is bound to comply with the Data Privacy Act of 2012 (RA 10173), its implementing Rules and Regulations and relevant issuances of the National Privacy Commission.

By participating in this video meeting/conference/webinar, you are consenting to our collection(including recording) in accordance with this Privacy Notice. Information is also processed via this video conferencing platform. Please refer to the privacy policy found in their website.

The information processed: name, email address, your image, video and audio will be used for attendance, documentation, communication and systems administration purposes.

Only authorised individuals of Department/Office/Unit organizing the event and other offices authorized will have access to this information and will not be disclosed to third parties without your approval.

We also use information gathered from you for abuse prevention and privacy protection.

DLS-CSB shall only retain the said personal information until it serves its purpose, after which it shall be securely disposed of.

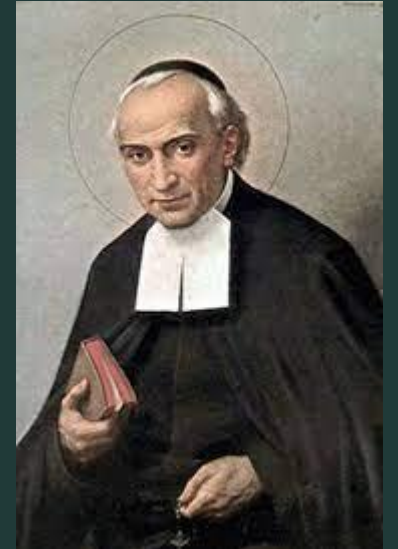
Queries and complaints can be directed to the Data Protection Officer at dpo@benilde.edu.ph.

Our Lasallian Prayer

I will continue, o my God, to do all my actions for the love of You.



*Mary, our Lady of the Star, pray for us.
Saint John-Baptiste de La Salle, pray for us.
Saint Benilde Romançon, pray for us.*



Live, Jesus, in our hearts, forever!

Lesson Outline

- Data Types
 - Numeric Data Types
 - Strings
 - Lists
 - Booleans

Numeric Data Types

Numeric Data Types

- Integers
- Floating Point Numbers
- Binary, Hexadecimal and Octal Numbers

Integers

- Numerical data types comprised of whole numbers
 - Uses the data type **int**
 - Can be positive or negative

```
>>> integer = 49
```

```
>>> negative_integer = -35
```

```
>>> print(type(integer), integer)
```

```
<class 'int'> 49
```

```
>>> print(type(negative_integer), negative_integer)
```

```
<class 'int'> -35
```

```
>>>
```

Integers

- Python integers have **unlimited precision** (unlike other languages)

```
>>> large_integer =  
34567898327463893216847532149022563647754227885439016662145553364327889985421  
  
>>> print(large_integer)  
  
34567898327463893216847532149022563647754227885439016662145553364327889985421  
  
>>>
```

Floating Point Numbers

- Numbers that can have decimal places
 - Uses the type **float**
 - You can force a number to have a floating point data type by using the **float()** function, i.e. **float(23)**

```
>>> n = 3.3333
```

```
>>> print(n)
```

```
3.3333
```

```
>>> import math
```

```
>>> print(type(math.pi), math.pi)
```

```
<class 'float'>, 3.141592653589793
```

```
>>> print(type(math.e), math.e)
```

```
<class 'float'>, 2.718281828459045
```

```
>>>
```

Alternative Number Systems (Binary, Hexadecimal and Octal)

- To write binary numbers, prefix it with **0b**
- To write hexadecimal numbers prefix it with **0x**
- To write octal numbers prefix it with **0o**

>>> 0b111

7

>>> 0x9ac

2476

>>> 0o200

128

Alternative Number Systems (Binary, Hexadecimal and Octal)

- To convert any decimal (base-10) number to binary, hexadecimal or octal numbers, use the built-in **bin()**, **hex()**, or **oct()** functions

```
>>> bin(7)
```

```
'0b111'
```

```
>>> hex(700)
```

```
'0x2bc'
```

```
>>> oct(70)
```

```
'0o106'
```

Arithmetic Operators

- You can perform arithmetic operations with numeric values/variables with the following operators
 - Note:** *The division of 2 numbers, regardless of their data types, will always yield a **floating point** number*

Operator	Result
$x + y$	Sum of x and y
$x - y$	Difference of x and y
$x * y$	Product of x and y
x / y	Quotient of x and y
$x // y$	Floored quotient of x and y
$x \% y$	Remainder of x and y
$-x$	x negated
$+x$	x unchanged
<code>abs(x)</code>	Absolute value or magnitude of x
<code>int(x)</code>	x converted to integer
<code>float(x)</code>	x converted to floating point
<code>divmod(x, y)</code>	Returns the pair (x // y, x % y)
<code>pow(x, y)</code>	x to the power y
$x ** y$	x to the power y

Assignment Operators

- Aside from the = simple assignment operator, Python has other assignment operators:

Operator	Example	Equivalent to
<code>+=</code>	<code>x += 7</code>	<code>x = x + 7</code>
<code>-=</code>	<code>x -= 7</code>	<code>x = x - 7</code>
<code>*=</code>	<code>x *= 7</code>	<code>x = x * 7</code>
<code>/=</code>	<code>x /= 7</code>	<code>x = x / 7</code>
<code>%=</code>	<code>x %= 7</code>	<code>x = x % 7</code>
<code>**=</code>	<code>x **= 7</code>	<code>x = x ** 7</code>
<code>//=</code>	<code>x //= 7</code>	<code>x = x // 7</code>

Order of Operations

- In Python (as with other languages), when combining different operators, there is an **order of operations** that take precedence:
 - Parentheses have the highest precedence
 - Next is the exponentiation operator (**)
 - Then, multiplication & division (including floor division and modulo)
 - Then, addition and subtraction
- Where two operators of the same precedence occurs, they are evaluated **left to right**
 - If this is not what you want, then *use parentheses* to alter the precedence

Pop Quiz

- How to write this equation in Python?

$$5(4 - 2) + \left(\frac{100}{\frac{5}{2}} \right) 2$$

Strings

Strings

- Strings are a sequence of characters, enclosed in single (') or double (") quotes.
- You can also build a ***multiline string*** by enclosing the characters in **triple quotes** ("""") or (""")
- You can use the ***** operator to repeat strings
- You can use the **+** operator to concatenate strings
 - Note however, in the **print()** function, you can use comma (,) to concatenate strings, but it adds a space between them.

Indexing

- Python strings can be indexed
 - The first character in the sequence is at index **0**
- You can extract a character from a string by using the `[]` syntax, with the index number inside the brackets
 - You can use a negative index to retrieve a character ***from the right end of the string***
 - Trying to use an index that is ***out-of-bounds*** will give an **IndexError**

0	1	2	3	4	5	6	7	8	9	10	11	12
P	y	t	h	o	n		i	s		f	u	n
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Slicing

- You can also get a slice/substring of a string by using the following format: *string[start_index : end_index]*
 - You can omit the start or end index, which means you are starting from the beginning or ending up to the end, respectively

```
>>> string = "championships"
```

```
>>> string[0:5]
```

```
'champ'
```

```
>>> string[5:9]
```

```
'ions'
```

```
>>> string[-5:-1]
```

```
'ship'
```

```
>>> string = "foobar"
```

```
>>> string[3:]
```

```
'bar'
```

```
>>> string[:3]
```

```
'foo'
```

Pop Quiz: String Slicing

- Given the following statements, predict what the output will be:

```
>>> "Living in a silent film"[5]
```

```
>>> "[8, 9, 10]"[4]
```

```
>>> "Don't try to frighten us with your sorcerer's ways"[13:19]
```

```
>>> "Testing 1, 2, 3"[7:]
```

```
>>> "A man, a plan, a canal: Panama."[:-1]
```

String Length

- You may use the **len()** function to determine the *length of the string*
 - An *empty string* ("") will have a length of 0

```
>>> question = "Who was the first Beatle to leave the group?"
```

```
>>> len(question)
```

```
44
```

String Formatting

- String formatting is important when you want to build new strings that are using existing values
- The most popular of these are:
 - String interpolation
 - The **str.format()** function
 - % formatting

String Interpolation

- ***String interpolation*** is the process of evaluating a string that has placeholders
 - Placeholders can hold expressions that yield a value, which is then placed inside a string
 - This is only available in **Python 3.6 and above**
- To interpolate, prefix the string quotes with **f**, and add a placeholder expression by enclosing them with **{ }** inside the string quotes:

```
>>> pie = 3.14
```

```
>>> f"I ate some {pie} and it was yummy!"
```

```
'I ate some 3.14 and it was yummy!'
```

str.format() Function

- Allows you to insert different values in positions within the string
 - This is similar to string interpolation, except you can't put expressions inside {}; you have to pass them inside the format() function:

```
>>> fruit = "bananas"
```

```
>>> "I love {}".format(fruit)
```

```
'I love bananas'
```

```
>>> age = 40
```

```
>>> years = 10
```

```
>>> string = "In {} years, I'll be {}"
```

```
>>> string.format(years, age)
```

```
"In 10 years I'll be 40"
```

% Formatting

- Similar to C language style formatting, use % format specifiers and specify the expressions to be substituted with the %

```
>>> number = 3
```

```
>>> pets = "cats"
```

```
>>> "They have %d %s" % (number, pets)
```

% Formatting

- Some basic % argument specifiers:
 - **%s** - String (or any object with a string representation, like numbers)
 - **%d** - Integers
 - **%f** - Floating point numbers
 - **%.<number of digits>f** - Floating point numbers with a fixed amount of digits to the right of the dot.
 - **%x/%X** - Integers in hex representation (lowercase/uppercase)

String Methods

- Aside from the **format()** method, there are also other string methods that can be used:
 - **str.capitalize()** – Capitalizes the first letter and makes the rest lowercase
 - **str.lower()** – Converts the string to lowercase
 - **str.upper()** – Converts the string to uppercase
 - **str.startswith()** – Checks whether the string starts with the specified prefix
 - **str.endswith()** – Checks whether the string ends with the specified suffix
 - **str.strip()** – Returns a copy of the string with the leading and trailing characters removed
 - **str.replace()** – Replaces all the occurrences of the old substring with the new substring
- More string methods are available at:
<https://docs.python.org/3/library/stdtypes.html#str>

Escape Sequences

- An escape sequence (starts with a backslash “\”) is a sequence of characters that does not represent its literal meaning when inside of a string:

```
>>> print("Hello\nWorld")
```

```
Hello
```

```
World
```

Escape Sequences

- List of escape sequences:

Escape Sequence	Definition
<code>\newline</code>	Backslash and newline ignored
<code>\\</code>	Backslash (\)
<code>\'</code>	Single quote (')
<code>\"</code>	Double quote (")
<code>\a</code>	ASCII Bell (BEL)
<code>\b</code>	ASCII Backspace (BS)
<code>\f</code>	ASCII Formfeed (FF)
<code>\n</code>	ASCII Linefeed (LF)
<code>\r</code>	ASCII Carriage Return (CR)
<code>\t</code>	ASCII Horizontal Tab (TAB)
<code>\v</code>	ASCII Vertical Tab (VT)
<code>\ooo</code>	Character with octal value ooo
<code>\xhh</code>	Character with hex value hh

Lists

Lists

- In Python, arrays (or the closest abstraction of them) are known as lists
 - Lists are an ***aggregate data type***, meaning that they are composed of other data types
 - Similar to strings, the values inside them are indexed, have a **length** property and a **count** of the objects inside of them
 - Python lists can hold values of different types (unlike C#, where arrays usually hold a single type only)
 - Python lists are also ***mutable***, meaning you can change the values inside of them, adding and removing items on the go

Lists

- Lists are made with its comma-separated elements enclosed in square brackets:

```
>>> digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

```
>>> digits
```

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

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

```
>>> letters
```

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

```
>>> mixed_list = [1, 3.14159, "Spring", "Summer", [1, 2, 3, 4]]
```

```
>>> mixed_list
```

```
[1, 3.14159, 'Spring', 'Summer', [1, 2, 3, 4]]
```

Lists

- To get the number of elements of the list, use the **len()** function:

```
>>> len(["a", "b", "c", "d"])
```

```
4
```

List Indexing

- To retrieve a particular element of the list, use the **square brackets** [] with the index number inside

- Negative indices can also be used, similar to strings

```
>>> fruits = ["apples", "bananas", "strawberries", "mangoes", "pears"]
```

```
>>> fruits[3]
```

```
'mangoes'
```

```
>>> fruits[-1]
```

```
'pears'
```


List Slicing

- Lists, like strings, can also be sliced
- Syntax is the same:
list[start_index : end_index]
- End index is also not included in the result
- Omitting the start_index or end_index will start or end at the first or last elements, respectively (same with strings)

```
>>> my_list = [10, 20, 30, 40, 50, 60, 70]
```

```
>>> my_list[4:5]
```

```
[50]
```

```
>>> my_list[5:]
```

```
[60, 70]
```

```
>>> my_list[:4]
```

```
[10, 20, 30, 40]
```

List Concatenation

- You can also add two lists together by using the `+` operator

```
>>> [1, 2, 3] + [4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

```
>>> ["a", "b", "c"] + [1, 2.0, 3]
```

```
['a', 'b', 'c', 1, 2.0, 3]
```

Changing Values in a List

- Since lists are mutable, you can change the value in a lists by using the assignment operator on a specific index:

```
>>> names = ["Eva", "Keziah", "John", "Diana"]
```

```
>>> names[2] = "Jean"
```

```
>>> names
```

```
['Eva', 'Keziah', 'Jean', 'Diana']
```

Adding Values to a List

- You can use the **list.append()** method to insert a value at the end of a list

```
>>> planets = ["Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus",  
"Neptune"]
```

```
>>> planets.append("Planet X")
```

```
>>> planets
```

```
['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', 'Planet X']
```

Assigning Values to a List Slice

- You can also assign slices of a list (when you use an index slice and change its values, Python allows changing only those elements)

```
>>> alphanumeric_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

```
>>> alphanumeric_list[4:7]
```

```
[5, 6, 7]
```

```
>>> alphanumeric_list[4:7] = ["a", "b", "c"]
```

```
>>> alphanumeric_list
```

```
[1, 2, 3, 4, 'a', 'b', 'c', 8, 9, 0]
```

Booleans

Booleans

- Boolean data types are values that can only be one of two values: **True** and **False**

```
>>> True
```

```
True
```

```
>>> False
```

```
False
```

```
>>> print(type(True), type(False))
```

```
<class 'bool'> <class 'bool'>
```

Comparison Operators

- Comparison operators compare values of objects or the objects' identities themselves
- Objects don't need to be of the same type
- Returns a Boolean value

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to
is	Object identity
is not	Negated object identity

Logical Operators

- Logical operators combine Boolean expressions in Python:

Operator	Result
not x	Returns false if x is true, else false
x and y	Returns x if x is false, else returns y
x or y	Returns y if x is false, else returns x

- **and** is a **short-circuit operator**; meaning, it only evaluates the second argument if the first one is **True**
- **or** is also a **short-circuit operator**; it only evaluates the second argument if the first one is **False**

Membership Operators

- The operators **in** and **not in** test for membership
 - All sequences (lists and strings) support this operator
- These operators go through each element to see whether the element being searched for is within or not within the list

```
>>> numbers = [1, 2, 3, 4, 5]
```

```
>>> 3 in numbers
```

```
True
```

```
>>> 100 in numbers
```

```
False
```

```
>>> sentence = "I like beef, mutton and pork"
```

```
>>> "chicken" not in sentence
```

```
True
```

```
>>> "beef" not in sentence
```

```
False
```



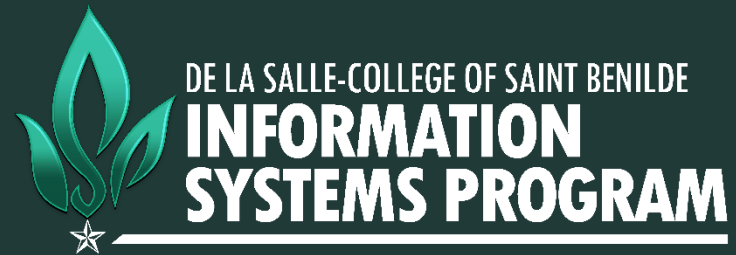
Pop Quiz

Replace ? with the correct Boolean operator:

- Consider the following code block:
n = 124
if n % 2 ? 0:
 print("Even")
- Consider the following code block:
age = 25
if age ? 18:
 print("Here is your legal pass.")
- Consider the following code block:
letter = "b"
if letter ? ["a", "e", "i", "o", "u"]:
 print(f"{letter}' is not a vowel.")

Application Development and Emerging Technologies (APPDAET)

Jino Raymundo



Data Privacy Notice

De La Salle-College of Saint Benilde (DLS-CSB) respects your right to privacy and is committed to protect the confidentiality of your personal information thus has adapted necessary organizational, technical and physical measures to secure it. DLS-CSB is bound to comply with the Data Privacy Act of 2012 (RA 10173), its implementing Rules and Regulations and relevant issuances of the National Privacy Commission.

By participating in this video meeting/conference/webinar, you are consenting to our collection(including recording) in accordance with this Privacy Notice. Information is also processed via this video conferencing platform. Please refer to the privacy policy found in their website.

The information processed: name, email address, your image, video and audio will be used for attendance, documentation, communication and systems administration purposes.

Only authorised individuals of Department/Office/Unit organizing the event and other offices authorized will have access to this information and will not be disclosed to third parties without your approval.

We also use information gathered from you for abuse prevention and privacy protection.

DLS-CSB shall only retain the said personal information until it serves its purpose, after which it shall be securely disposed of.

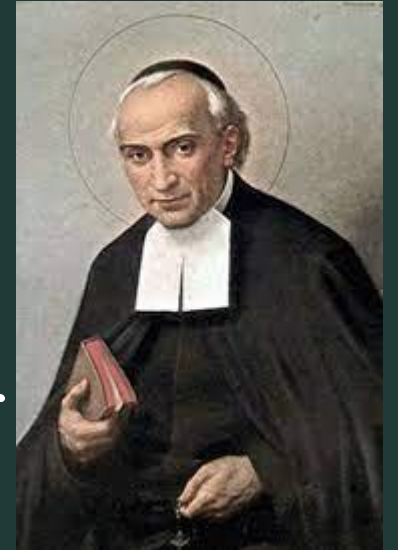
Queries and complaints can be directed to the Data Protection Officer at dpo@benilde.edu.ph.

Our Lasallian Prayer

I will continue, o my God, to do all my actions for the love of You.



*Mary, our Lady of the Star, pray for us.
Saint John-Baptiste de La Salle, pray for us.
Saint Benilde Romançon, pray for us.*



Live, Jesus, in our hearts, forever!

Lesson Outline

- Control Statements
 - The if Statement
 - The while Statement
 - The for Loop
 - The range Function
 - Breaking Out of Loops

Control Statements

The if Statement

- Executes a block of code when the condition is **True**; otherwise, it can run an alternative block of code in its **else** clause
 - The **else** clause is optional
 - You can chain multiple **if** statements by using **elif** (else if)

if *condition*:

run this code if the condition evaluates to True

elif *condition_2*:

run this code if condition_2 evaluates to True (and first condition evaluated to False)

.

.

else:

run this code if all above conditions evaluate to False

The while Statement

- Allows you to execute a block of code repeatedly, as long as the condition remains True
 - Can also have an **else** clause that will be executed exactly once when the condition that's mentioned is no longer true

while *condition:*

Run this code while condition is true

Replace the "condition" above with an actual condition

This code keeps running as long as the condition evaluates to True

else:

Run the code in here once the condition is no longer true

This code only runs one time unlike the code in the while block

The for Loop

- Used when you have a block of code that you would like to execute repeatedly a given number of times
 - Works for *iterables* (such as strings, lists, dictionaries and files)
 - Can also have an **else** statement that executed exactly once when the loop exits cleanly

for *member in iterable:*

Execute this code for each constituent member of the iterable

else:

Execute this code after the last loop has been executed

The range Function

- The **range** function is a built-in function that generates a list of numbers
 - Mostly used to iterate over using a **for** loop

range(*[start]*, *stop*, *[step]*)

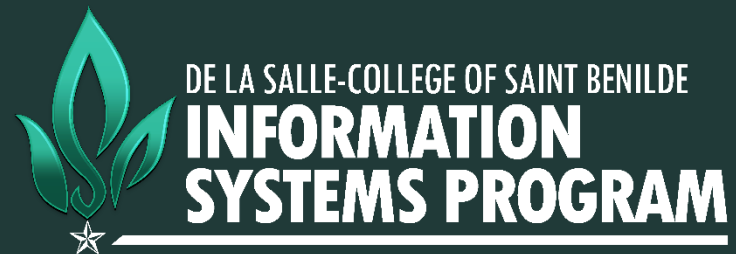
- **start**: This is the starting number of the sequence.
- **stop**: This means generate numbers up to but not including this number.
- **step**: This is the difference between each number in the sequence.

Breaking Out of Loops

- Python provides 3 statements that can be used to break out of a loop:
 - **break** – exits the current loop
 - **continue** – restarts the loop
 - **pass** – allows you to handle an external trigger condition without affecting the execution of the loop

Application Development and Emerging Technologies (APPDAET)

Jino Raymundo



Data Privacy Notice

De La Salle-College of Saint Benilde (DLS-CSB) respects your right to privacy and is committed to protect the confidentiality of your personal information thus has adapted necessary organizational, technical and physical measures to secure it. DLS-CSB is bound to comply with the Data Privacy Act of 2012 (RA 10173), its implementing Rules and Regulations and relevant issuances of the National Privacy Commission.

By participating in this video meeting/conference/webinar, you are consenting to our collection(including recording) in accordance with this Privacy Notice. Information is also processed via this video conferencing platform. Please refer to the privacy policy found in their website.

The information processed: name, email address, your image, video and audio will be used for attendance, documentation, communication and systems administration purposes.

Only authorised individuals of Department/Office/Unit organizing the event and other offices authorized will have access to this information and will not be disclosed to third parties without your approval.

We also use information gathered from you for abuse prevention and privacy protection.

DLS-CSB shall only retain the said personal information until it serves its purpose, after which it shall be securely disposed of.

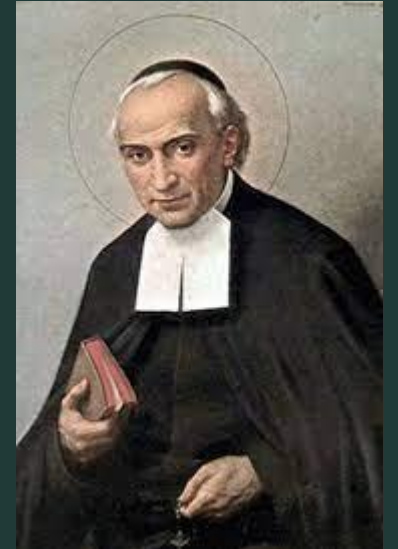
Queries and complaints can be directed to the Data Protection Officer at dpo@benilde.edu.ph.

Our Lasallian Prayer

I will continue, o my God, to do all my actions for the love of You.



*Mary, our Lady of the Star, pray for us.
Saint John-Baptiste de La Salle, pray for us.
Saint Benilde Romançon, pray for us.*



Live, Jesus, in our hearts, forever!

Lesson Outline

- Functions

Functions

Built-in Functions

- The Python interpreter has a number of built-in functions and types that are always available:
 - **input([prompt])** - This optionally prints the prompt to the terminal. It then reads a line from the input and returns that line
 - **print()** - Prints objects to the text stream file or the terminal
 - **map()** - Returns an iterator that applies a function to every item of the iterable, yielding the results

User-Defined Functions

- You can also create your own functions using the **def** construct:

```
def function_name( parameter_one, parameter_two, parameter_n ):
    # Logic goes here
    return
```

Global and Local Variables

- Variables that are declared inside a function body is called a **local variable**
 - These variables are only accessible inside the function (also known as a ***local scope***)
- Variables that are declared outside a function body is called a **global variable**
 - These variables can be accessible inside or outside functions (also known as a ***global scope***)

The return Statement

- The **return** statement is used to return a value from a function
 - Without a **return** statement, the function will return **None**

Function Arguments

- Python supports several types of function arguments:
 - Required arguments
 - Keyword arguments
 - Default arguments
 - Variable arguments

Required Arguments

- Required arguments are types of arguments that have to be present when calling a function
 - This is the default type of argument when creating a user-defined function
 - Required arguments need to be followed according to the order it is declared:

```
def division(first, second):  
    return first/second
```

```
x = division(10, 2)
```


Keyword Arguments

- Sometimes, you might want to specify the name of the argument as part of your parameter passing, in order to use arguments out-of-order:

```
def division(first, second):  
    return first/second
```

```
quotient = division(second=2, first=10)  
print(quotient)
```

Default Arguments

- Some parameters can have a default value if the argument is not supplied:

```
def division(first, second=2):  
    return first/second
```

```
quotient = division(10)  
print(quotient)
```

Variable Arguments

- You can also create a function with a **variable number of arguments** using the ***** construct:

```
def addition(*args):  
    total = 0  
    for i in args:  
        total += i  
    return total
```

```
answer = addition(20, 10, 5, 1)  
print(answer)
```

Anonymous Functions

- Anonymous functions are also known as **lambda functions**
 - They are called anonymous because they don't need to be named in their definition
 - These functions are usually *throwaway*, meaning they are only required where they are defined, and are not to be called in other parts of the code
 - Uses the keyword **lambda**

lambda *argument_list: expression*

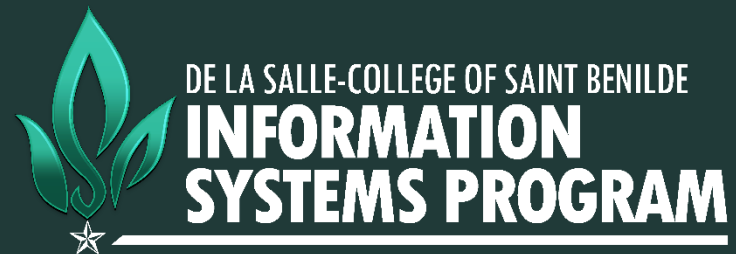
Using Anonymous Functions

- They are often used in combination with the **map()**, **reduce()** and **filter()** functions:

```
numbers = [2, 4, 6, 8, 10]  
squared = map(lambda num: num ** 2, numbers)
```

Application Development and Emerging Technologies (APPDAET)

Jino Raymundo



Data Privacy Notice

De La Salle-College of Saint Benilde (DLS-CSB) respects your right to privacy and is committed to protect the confidentiality of your personal information thus has adapted necessary organizational, technical and physical measures to secure it. DLS-CSB is bound to comply with the Data Privacy Act of 2012 (RA 10173), its implementing Rules and Regulations and relevant issuances of the National Privacy Commission.

By participating in this video meeting/conference/webinar, you are consenting to our collection(including recording) in accordance with this Privacy Notice. Information is also processed via this video conferencing platform. Please refer to the privacy policy found in their website.

The information processed: name, email address, your image, video and audio will be used for attendance, documentation, communication and systems administration purposes.

Only authorised individuals of Department/Office/Unit organizing the event and other offices authorized will have access to this information and will not be disclosed to third parties without your approval.

We also use information gathered from you for abuse prevention and privacy protection.

DLS-CSB shall only retain the said personal information until it serves its purpose, after which it shall be securely disposed of.

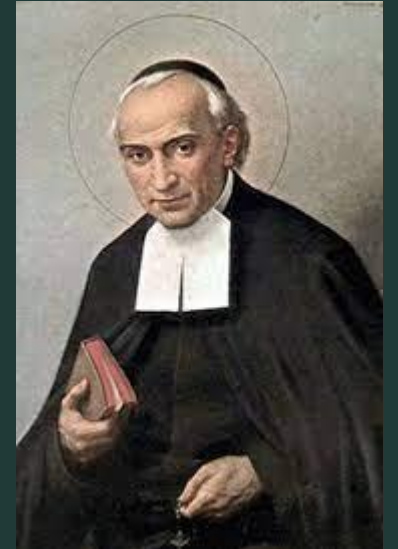
Queries and complaints can be directed to the Data Protection Officer at dpo@benilde.edu.ph.

Our Lasallian Prayer

I will continue, o my God, to do all my actions for the love of You.



*Mary, our Lady of the Star, pray for us.
Saint John-Baptiste de La Salle, pray for us.
Saint Benilde Romançon, pray for us.*



Live, Jesus, in our hearts, forever!

Lesson Outline

- Lists and Tuples

Lists and Tuples

Lists vs. Tuples

- A **list** is a data structure that holds *ordered collections* of related data
 - Also known as *arrays* in other programming languages
 - Lists in Python are more flexible and powerful than arrays:
 - They are ordered
 - They contain objects of arbitrary types
 - The elements of a list can be accessed by an index
 - They are arbitrarily nestable (can contain other lists as sublists)
 - They have variable sizes
 - They are **mutable** (you can change the contents of a list)

Lists vs. Tuples

- **Tuples** are used to hold together multiple related objects
 - They are similar to lists, but differ in that they don't have all the functionality afforded by lists
 - The key difference between lists and tuples is the **immutability** of tuples (i.e. you cannot change the elements of a tuple once they are set)

List Syntax

```
# Empty list
```

```
[]
```

```
# List containing numbers
```

```
[2, 4, 6, 8, 10]
```

```
# List with mixed types
```

```
["one", 2, "three", ["five", 6]]
```

List Methods

- `list.append(item)`
- `list.extend(iterable)`
- `list.insert(index, item)`
- `list.remove(item)`
- `list.pop([index])`
- `list.clear()`
- `list.index(item [, start [, end]])`
- `list.count(item)`
- `list.sort(key=None, reverse=False)`
- `list.reverse()`
- `list.copy()`

List Comprehensions

- List comprehensions are a feature of Python that give us a clean, concise way to create lists
 - A common use case would be when you need to create a list where each element is the result of some operations applied to each member of another sequence or iterable object
 - Syntax: **[*expression* **for** *list_var* **in** *iterable* **if** *condition*]**

```
>>> squares = [num**2 for num in range(1, 11)]  
>>> squares  
=> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
>>> squares = [num**2 for num in range(1, 11) if num%2 == 0]  
>>> squares  
=> [4, 16, 36, 64, 100]
```

Tuple Syntax

- Main advantages of using tuples:
 - They are better suited for use with different (heterogeneous) data types
 - Can be used as a key for a dictionary (to be discussed next week) due to the immutability of tuples
 - Iterating over tuples is much faster than lists
 - They are better for passing around data that you don't want changed
- Tuples can be created by placing all comma-separated values in **parentheses ()**
 - You can create a tuple by simply assigning a comma-separated list; but when outputting them, it automatically adds the parentheses
 - You cannot have a tuple with only 1 element
 - You can add a comma, followed by a blank item; i.e. ('thing',)

```
pets = ('dog', 'cat', 'parrot')  
pets  
=> ('dog', 'cat', 'parrot')
```

```
type(pets)  
=> <class 'tuple'>
```

```
pets2 = 'dog', 'cat', 'parrot'  
pets2  
=> ('dog', 'cat', 'parrot')
```

```
type(pets2)  
=> <class 'tuple'>
```


Casting Lists to Tuples

- You can convert a list into a tuple by using the **tuple()** method.
 - Inside the tuple() method, you can also do list comprehensions

```
numbers = tuple(range(1,11))  
numbers  
=> (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
numbers = tuple([i for i in range(1,11)])  
numbers  
=> (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Accessing Tuple Elements

- **Indexing**

- Similar to lists, use the index operator `[]` to access an element in a tuple using its index.
- Tuple indices also start at 0.
- Negative indices will start from the end of the tuple

- **Slicing**

- Tuple slicing syntax:
tuple_var[start_index, stop_index, increment]
 - **Start index:** The index at which to start the slicing; the start index is included in the slice
 - **Stop index:** The index at which to stop slicing; the stop index is not included in the slice
 - **Increment:** Determines how many steps to take in the tuple when creating the slice.
- Start index, stop index and increment can all be optional (just make sure to put the appropriate colon)

```
numbers = tuple([i for i in range(1,11)])  
numbers  
=> (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
even = numbers[1::2]  
even  
=> (2, 4, 6, 8, 10)
```

Tuple Methods

- **any()**: used to discover whether any element of a tuple is an iterable
- **count()**: returns the number of occurrences of an item in a tuple
- **min()**: Returns the smallest element in a tuple
- **max()**: Returns the largest element in a tuple
- **len()**: Returns the total number of elements in a tuple
- Tuples can also be ***concatenated*** using the + operator