Python Programming 1

Unit 1 Module 1

Section 1 Getting Started with Python in Jupyter Notebooks

- # comments
- Print function
  Syntax: print()
  print("Hello World")
  print(varName)
  print(9)

Section 2 Types & Variables

- Variables – placeholder – used to store values
  - String
  - Integer – whole numbers
  - Float – decimals
- Assignment
  - Giving a variable a value
    varName = value
    Message = "Hello"
    Age = 16
    gpa = 3.2
  - Num1 does NOT equal num1
  - Can't add strings and numbers
- Concatenate Strings
  NewMessage = Message1 + Message2
- Variable reassignment
  num = 12
  num = 9

Section 3 type() Function

- type() Function
  - returns the data type of the given data or variable
  - Examples
    type(9) returns int for Integer
    type("9") returns str for string

Section 4 Addition & Errors

- Errors
  - Syntax
    - Rules of language error

    - RunTime
        - Error that causes program to crash
        - Divide by 0
    - Logic
        - Unexpected results
- Errors in Python
    - TypeError
        - Illegal type operation used
        - Adding string and int
    - SyntaxError
        - Unknown code used
    - NameError
        - Unknown command

Section 5 ACSII Art

- Ascii art

Section 6 Input

- User Input
    - input() function
    - Example
      input("Put a question or prompt here")
    - Should be on the right side of an equals sign
      var = input("Prompt/Question")
    - Always returns a string data type

Section 7 Print Formatting

- In PowerPoint
    - Type Conversions
        - Casting
        - Example
          gradeEntered = input("Enter grade")
          grade = int(gradeEntered)
- Printing numbers with text
    - Must cast the numeric variable to string
    - Example:
      print("Grade: " + str(grade))
- Can concatenate (merge) with , instead of + in the print function
    - Can also combine , with +
    - Adds a space between

Section 8 Quote Display & Boolean

- Using a quote (" ') in a string
    - Use " for the quotes you want to show and ' for the quotes around the entire string

    - Example
      print('This is a "string"')
- Boolean
    - True/False
    - Functions to test strings that return true/false
      isalpha() – A-Z or a-z
      isalnum() - at least 1 character and all are alphabetical or digits
      istitle() – All words are capitalized
      isdigit() – All digits
      islower() – lowercase
      isupper() – uppercase
      startswith() – starts with a certain character

Section 9 String Formatting & the 'in' Keyword

- String Formatting
    - .capitalize() - capitalizes the first character of a string
    - .lower() - all characters of a string are made lowercase
    - .upper() - all characters of a string are made uppercase
    - .swapcase() - all characters of a string are made to switch case
      upper becomes lower and vice versa
    - .title() - each 'word' separated by a space is capitalized
- Combining Functions
    - Can nest functions
    - Example
      message = input("Enter name").upper())
- in Function
    - returns true if searched text is in string
    - Example
      message = input("Enter some text")
      print ("Text" in message)
        - If the "Text" is in the message then it returns true

Unit 1 Module 2 Functions

Section 1 Simple Functions

- Built-in Function – Provided by Python to use
    - Example: print()
- Parameter - define the variable Arguments that can be passed to the Function.
    - Part of function definition
    - Become variables to be used in the function code
- Argument – actual variable/values passed to the function
    - Part of call statement
- Keyword to start a function: def
- A function name **starts with a letter** or underscore (usually a lowercase letter).

- Function names can contain **letters, numbers or underscores**.
- Parentheses **()** follow the function name.
- A colon **:** follows the parenthesis.
- The code for the function is indented under the function definition (use 4 spaces for this course).
- Call a simple function using the function name followed by parentheses.
- Defining function parameters¶
    - Parameters are defined inside of the parentheses as part of a function def statement
    - Parameters are typically copies of objects that are available for use in function code

```
def say_this(phrase):
    print(phrase)
```

    - Functions can have default arguments
    - Default arguments are used if no argument is supplied

Section 2: Function return and Multi-Parameters

- **`return`** keyword in a function *returns* a value after *exiting* the function.

Example:

```
# Message double returns the string Argument doubled
def msg_double(phrase):
      double = phrase + " " + phrase
      return double

# save return value in variable
msg_2x = msg_double("let's go")
print(msg_2x)
```

- Functions can have multiple parameters separated by commas.
  Example:

```
def make_schedule(period1, period2):
    schedule = ("[1st] " + period1.title() + ", [2nd] " + period2.ti
tle())
    return schedule

student_schedule = make_schedule("mathematics", "history")
print("SCHEDULE:", student_schedule)
```

Section 3: Functions, Arguments, & Parameters

- In programming, **sequence** refers to the order that code is processed. Objects in Python, such as variables and functions, are not available until they have been processed.
- Processing sequence flows from the top of a page of code to the bottom. This often means that Function definitions are placed at the beginning of a page of code.
- A programming best practice is to avoid hard-coding values when possible.
    - Use variables and verse hard-coded
    - Often preferable to use input such as a configuration file (advanced topic) or user input. These practices allow changing the data without disturbing the main code and makes code more reusable.

Unit 1 Module 3: Conditionals

Section 1 Boolean Conditionals & if/else

- Concept: Boolean Conditional
    - Conditionals use True or False
- Syntax:
  if *some_conditional*
  #do_this
  else
  #do_this

- Example

```
hot_tea = True

if hot_tea:
    print("enjoy some hot tea!")
else:
    print("enjoy some tea, and perhaps try hot tea next time.")
```

- Note the else is not required

Section 2: Comparison Operators

- Concept: Comparison Operators
  >
  <
  >=, <=
  ==
  Assign = vs compare ==
  !=
- Examples

```
x = 21
if x > 25:
    print("x is already bigger than 25")
else:
    print("x was", x)
    x = 25

x = 18
if x + 18 == x + x:
    print("Pass: x + 18 is equal to", x + x)
else:
    print("Fail: x + 18 is not equal to", x + x)
```

```
# Note "!" means "not"
x = 18
test_value = 18
if x != test_value:
    print('x is not', test_value)
else:
    print('x is', test_value)
```

Section 3: Conditionals – String Comparisons

- Strings can be equal == or unequal !=
- Strings can be greater than > or less than <
- Alphabetically "A" is less than "B"
- Lowercase "a" is greater than uppercase "A"
- Example

```
msg = "Save the notebook"

if msg.lower() == "save the notebook":
    print("message as expected")
else:
    print("message not as expected")
```

Section 4: Conditionals, elif, and Casting

- Review
  - if means "if a condition exists then do some task." if is usually followed by else.
  - else means "or else after we have tested if, then do an alternative task".
  - When there is a need to test for multiple conditions there is elif.
  - elif statement follows if, and means "else, if " another condition exists do something else
  - elif can be used many times
  - else is used after the last test condition (if or elif)
- Casting
  - Casting is the conversion from one data type to another, such as converting from `str` to `int`.
- int() Function
  - The int() function can convert strings that represent whole counting numbers into integers and strip decimals to convert float numbers to integers.
  - int("1") = 1 the string representing the integer character "1", cast to a number
  - int(5.1) = 5 the decimal (float), 5.1, truncated into a non-decimal (integer)
  - int("5.1") = ValueError "5.1" isn't a string representation of integer, int() can cast only strings representing integer values

Section 5: Conditionals, Type, and Mathematics Extended

- Basic Math operators
  +, -, *, /

Unit 1 Module 4: Nested if and while

Section 1 Nested Conditionals

- Syntax:

```
if
    if
    else
else
```

Section 2 Printing with the () escape sequence

- Escape Sequences
  - Escape sequences all start with a backslash (\)
  - Escape sequences can be used to display characters in Python reserved for formatting
    \\ Backslash (\)
    \' Single quote (')
    \" Double quote (")
  - Escape sequences are part of special formatting charcters\n Linefeed
    \t Tab
    \n return or newline
- We use escape sequences in strings - usually with print() statements.
- Example:

```
print('Hello World!\nI am formatting print ')
```

Section 3 while() Loops and Incrementing

- `while True:`
  - is known as the **forever loop** because it ...loops forever
  - Using the **`while True:`** statement results in a loop that continues to run forever ...or, until the loop is interrupted, such as with a **`break`** statement.
- break
  - causes code flow to exit the loop
  - a **conditional** can implement **`break`** to exit a **`while True`** loop
  - `while True` loops forever unless a `break` statement is used

- Incrementing
  - Note: this can be used as a counter in your code
  - num = num + 1
    num += 1
  - Adds 1 to the variable num
- Decrementing
  - num = num – 1
    num -+ 1
  - Subtracts 1 to the variable num

Section 4 while() Boolean Loops

- `While` with Boolean comparison operator
- `while ():`
    - With increment we use break when count becomes greater than some number.
    - The same result can be achieved by using `while x < 5:.`
- Using `while` with a Boolean String Tests
    - A while loop can be controlled by Boolean strings such as `while name.isalpha() == False:`