Python Notes on CS150B class taught by my professor Elisa Cundiff.

READING 0

- Moore's Law - the trend of reducing the size of a switch by two.
- Agricultural age lasted long, and the industrial age brought population explosion. The information age is still making enormous differences to our lives dramatically.
- Joseph Marie Jacquard creator of Loom silk weaving machine that can take different inputs (punched cards) to produce different outputs)
- Charles Babbage - creator of Difference Engine and the later successor Analytical Engine.
- Ada Lovelace - the first computer programmer, augmented Charles Babbage's document about Analytical Engine with her notes (A-G), tripling the size. This Note G is considered as the first computer program.
- Herman Hollerith - sold punched cards and helped the U.S Census which used the punched cards to store American citizens' data. Then, he opened Computing-Tabulating Recording Company which was later named as International Business Machines Corporation (IBM).
- Alan Turing - helped Allies win WW 2 by deciphering the intercepted messages using his cryptanalytic machine to break the German Enigma code. He also introduced the Turing Test to evaluate the intelligence of a computer.
- Grace Hopper- credited as the creator of the first compiler. Created the COBOL programming language used in business context: still used but criticized for being too verbose. She and her staff first introduced the term, Computer bug''.
- A single 0 and 1 is called a ***bit,*** and 8 bits altogether is called a ***byte***.
- ASCII and Unicode - are character sets.
- 2 to the power 30 = about 1 billion = 1Gb (1 gigabit) or IEC prefix 1 Gibibit
- 2 to the power 10 = about a thousand = 1Kb or in IEC prefix 1Kib
- 2 to the power of 16 = 64 Kb (kilo bit) 64 Kib (kibibit)
- 2 to the power of 20 = 1Mb (megabit) or in IEC prefix 1Mebi (mebibit)

Reading 1

- A **statement** is a program instruction.
- **Expressions** are pieces of code that return a value.
- **Variables** are named references that represent the object.
- In python, to print a value, **print( ) built-in** function is used.
- Single-line commenting is done by putting a hash (pound) symbol: #
- anything commented is not noticed by the interpreter.
- Python Interpreter reads the code **line-by-line** in **top-down** order.

- In python, strangely enough, you don't need to define the type of the variable like you did in C++ or JS or any other programming language.

- age = 2037 - 2003

Above, firstly the right side expression is evaluated and then the prevailing values is stored Into the **age** variable. Note that there is no const or let (JS) or int (C++) data types written next to the **age** variable.

- There are two ways of printing a value to the screen:

  1) print("Welcome dear user,", userName)

Above, there is no space left after the comma, but with **comma** (,) concatenation, there will be added an extra space to wield the left and right side values with a space in between. Another useful thing that the comma concatenation does is that it is able to concatenate both string and integer type values without any problem.

  2) print("Welcome dear user, " + userName)

Above, we added an extra space after the comma since the **+ concatenation** does not put an extra space between left and right side values like the **comma concatenation** did. Note that with + concatenation you cannot add a string and integer values types as you will get a TypeError. To add a string and an integer, convert the integer into string without mutating the integer using the built-in ***str( )*** function which takes in an integer value and converts it into string.

- *print("You are " + str(userAge) +" year(s) old") → You are 12 year(s) old.*

- Text enclosed in quotes is known as a **string literal.**
- Each user of print( ) starts on a new line.
- As has been said, each print() function call leaves a newline after being executed. But to prevent that from happening, we can add an attribute called **end** and set it equal to whatever we want which will then replace the newline character (\n).

- **Default**: print("My name is:", end ='\n') → My name is
  print("Sanjar", end ="\n") Sanjar
- **A space**: print("My name is:", end=' ') → My name is Sanjar.
  print(" Sanjar")

- An empty print( ) can be used to add a newline.
- Whitespaces are a space, tabs, newlines or any other spacing characters.
- **input()** built-in function is used to get an input from the user.
- The input() function only returns a stringified value and if you are going to use the values returned by the input() as a number to do some math, you need to convert the string into an integer type using the **int( )** built-in function.

- userAge = int(input("Enter your age: "))

- Above we wrapped the input function inside the int function, and that is perfectly feasible since the interpreter first evaluates the input and then converts it into an integer type. Note that you can pass a prompt string to ask for a value from the user.
- Putting multiple **print()** statements on the same line is a **Syntax error**.
- Note that in case there is a Syntax Error in the program, none of the valid **print()** statements execute unless the error is corrected.
- **Runtime Errors:** multiplying by strings, or dividing by 0
- Abrupt and sudden termination of a program is called a program crash caused by a runtime error.

Error type	Description
SyntaxError	The program contains invalid code that cannot be understood.
IndentationError	The lines of the program are not properly indented.
ValueError	An invalid value is used – can occur if giving letters to int().
NameError	The program tries to use a variable that does not exist.
TypeError	An operation uses incorrect types – can occur if adding an integer to a strin

- Logic Errors: errors that you did not expect to happen like 2*4 where 4 should have been 40 instead. That results in a bug.
- In the 1960s and 70s scripting languages got popular. A script is a program whose instructions are executed by an interpreter
- Guido Van Rossum created Python (simplicity and readability) in the 1980s
- Python 3.0 is not **Backwards compatible.**
- Python is an open-source language, and any corrections by the community is welcome.
- Since \ is an escaping character but you need to use it anyway, then you should write double \\ to escape the \ .
- Added to str( ) and int( ), we have got float( ) which converts passed value into float.

READING 2

- Incrementing means adding one to the current variable: x = x + 1
- The identifier or the name of a variable can be made up of alphabetic characters, digits, and underscores as long as the digits are never the first character of the variable name.
- Python is case sensitive meaning that dog != Dog
- Valid names:

c, cat, b1, _dog, Age

- Invalid identifiers: sanja£r (no symbol allowed except for an underscore), 4b (no digits as the fist character), user Name (no space in between).
- You can use __identifier__ but these double opening underscores and closing double underscores have special meaning, so it is better to avoid using them for now.
- The names of the identifiers must be informative and should tell the reader what task they are going to do. EG: age, temperatureCelcius, userName as so on.
- Identifier naming conventions:

1) Underscore Convention: all words are joined with underscores and are lower cased my_age, user_name, country_currency, farenheit_to_celcius etc.

2) Camel Case Convention: the first alpha character is lower but others not.
myName, yourCountry, employeeSalary, celsiusToFarenheit etc.

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

- There are some key- words that are reserved by the interpreter shown on the left.

- int + int = int
- float + int or float + float = float
- int to float conversion: 25 becomes 25.0
- float to int conversion: 25.9 becomes 25, dropping the fraction part
- To raise a number x to the power y, we write it as x**y
- Surprisingly, floating point numbers can store as small numbers as 2.3*(10 to the power of -308) or a big number as 18*(10 to the power of 308)
- An overflow occurs in case you create a variable and try to store a bigger value than the amount possible.
- Sometimes you want to print only a certain number of digits after the . in a floating point number. EG: printing only two digits after the floating point.

print(f" { myFloat: .2f } ") → imagine my float was 2.34345, this would print 2.34
print(f" { 2.346574: .2f } ") → this would produce 2.35.

- You can use the PI (3.14159...) that is mostly used in mathematics by importing the math module and calling **pi** property on imported **math.**

  EG: import math
  print(f " {math.pi: .4f} ") → 3.1416

- Precedence rules say which expression should first be evaluated. Top-down order of precedence: **Brackets, exponent, unary (- negation), * and /, + and -**, in other cases **left to right** order is kept.
- A good practice is to use a space around each operator for code readability. But, in the case of - unary operator, we do not have to use it since otherwise it would be thought of as a subtraction operator but not negation operator.
- Compound operators or shorthand operators:
  Imagine x = 3;
  x += 2; → x = x + 2
  x -= 2; → x = x - 2
  x *= 2; → x = x * 2
  x /= 2; → x = x / 2

x %= 2; → x = x % 2

print(x) → 1 after all these expressions it would yield 1.

- Note that in Python, there is not x-- or x++, or x.
- Do not try to write numbers with commas in between. 3,000 is written as 3000.
- Function is a useful block containing statements that perform a task. Functions help us a lot since we will not have to write the same code over and over again.
- Function definition is the head of a function: *def func( ):*

your function can have parameters inside its braces and you need to use the **def** keyword which stands for **define**, and you also need to put a colon after the closing brace.

- A function can only return a value at a time but if you want to return multiple values at once, you could pack them in a tuple of a list and then return it and then unpack it in the calling function.
- A function without any return keyword return None by default.
- A parameter is in the function definition but argument is given in the function's call.
- Unusually, Python does not allow for **Switch control structure cases**.

STRINGS

- Strings are sequences of characters. Strings are SEQUENCE TYPES which means they represent an order in the elements. A string ,, Sam'' has characters at 0, 1, and 2 indices.
- To find the length (number of characters in a string) of a string, we use the built-in function **len( )** and pass our string in it.
  - print(len("Sanjar")) → 6 Note that the counting length starts at 1.
- If your string is too long and you wish to write some part of it on a new line, then you don't need to write multiple print statements with end=' ' at the end each time. Instead use the \ escaping character to define you are simply going a line down without adding no space.

```
myString = "His "\
    "Your " \
    "prince "\
    "is nothing"

main
C:\Users\Fresh\pythonProjec
His Your prince is nothing
```

- You are able to access any single character in a string by appending a pair of square brackets [ ] right next to the string name, and passing in the index at which the character you are after is at.
- Left-to-Right indexing starts at **0** till **len(string)-1** but Right-to-Left indexing starts at **-1.**
- Note that it is impossible to mutate characters of a string, if you do so you will get a TypeError message. BUT, you can still update the whole string.

```
userName = "Sanjar"
userName[0] = 'K'

main
C:\Users\Fresh\pythonProject\venv\Scripts\python.exe C:/U
Traceback (most recent call last):
  File "C:\Users\Fresh\pythonProject\main.py", line 2, in
    userName[0] = 'K'
TypeError: 'str' object does not support item assignment
```

- To concatenate two or more strings, we can use + operator. Note that concatenation does not mutate any of the strings involved. It

simply creates a new string which you, if you are smart, need to save into a variable and then use.

- - Example: myString = “Sanjar” + “Yuldashev” → “SanjarYuldashev”
- In Python, we have formatted string literals that make your job easy in creating strings with a mix of variables, without having to use , or + to concatenate or to worry about casting of int into string.
- - Example: userComments = f” My name’s {userName}, I’m a {age} year old {job}.”
  print(useComments) → *My name’s Sanjar, I’m a 19 year old student.*
- These { } of f-string is called a place holder or replacement field since they allow the expressions inside of them to produce a value first.
- One of the additional features of f-string is that you can print both the expression itself and the result:
- Examples: - print(f ”{2**3=}”) → 2**3 = 8
- print(f”{ {2**3=}}”) → two layers give the string. → {2**3=}
- print(f”{{{2**3 = }}}”) → {2**3 = 8}
- userAge = 18
- print(f“{userAge = }”) → userAge = 18
- Another useful tool that we have is the Format **specification** which allows us to put trailing zeros in front of a number, represent a number as a floating number, and align it to left and right. Basically, the syntax is the *value itself on the left* **:** *how the value is formatted on the right*.
- *Example:* { 4: .2f} → 4.00: The .2f is a **presentation type**. There are other types of presentation types that we can use.
- **Presentation Types**:
- By default: print(f” { “Adan”: s}”) → Adan: this is a string and anything put inside f “{}” by default turned into string.
- For only integers: *print(f”{12: d}”)* → 12. Note, in case you use a floating number, you will get an error.
- For only integers: *print(f”{ 7: b}”)* → *111*
- For only integers: x is for lower letters, X for upper. *print(f”{31: x}”)* → *1f*
- Exponent Notation: *print(f”{44: e}”)* → *4.400000e01*
- f : only six places of precision: *print(f”{23.31: f}”)* → *23.310000*
- Precision defined: *print(f“{23.53245: .3f}”)* → *23.532*
- Leading 0 notation, only INTEGERS: *print(f”{12: 03d}”)* → *012. Note that in the presentation part, you need to first write 0 to denote that you are going to put leading zeros and then write how many rooms there should be altogether considering the actual number too, then write ‘d’*
- In case there are more digits already then there will not be any leading zeros at all.

- *print(f"{123: 03d}") → 123 as you might see there is no leading zero at all.*

- ***IF ELSE STATEMENTS***
- We use == for equality evaluation: *userAge = 12, then userAge == 12* → True
- Similarly, inequality sign is denoted by **!=.** It gives True whenever x != 12, x and 12 are different numbers.
- The syntax for the conditional branching is as follows:
- if age == 12:
    pass
- elif age == 13:
    pass
  else:
    pass
- Operators that are used in branching are >=, <=, <, >, != and ==
- Operator chaining in Python: ***x<y<z*** here, the evaluation starts from the left to the right, and firstly x<y is evaluated, if true y<z is evaluated, but if x<y fails, then the comparison stops and False is immediately returned.

- ***READING FILES***

myJournal = open ("journal.txt") → open function creates a file object
contents = myjournal.read() → read function saves the contents of the file as a string.
- open function needs a path to the file, you can also pass the whole path to make the directory more clear.
- file. close( ) → literally closes the file after which writing and reading from the file is not possible.
- There are three ways of reading the contents of a file. The first is file.read() which simply reads the contents the same as they appear and makes them a string. The second is file. readlines() which returns a list whose first element is a string representing the first line, the second line string being stored at index 1.

```
['Today\n', 'my professor of\n', 'MicroEconomics\n', 'told me about\n']
```

- We can also tell the interpreter how much amount of data to read by passing the byte value, say 500, so that we don't read too much.
- When EOF (end of file) is reached, reading stops.
- The third type is file. readline() which only read a line on a call. But, to get more, we have to call more of this function. It is better since you will have more control over how much

data to read from your file.

```
"""Echo the contents of a file."""
f = open('myfile.txt')

for line in f:
    print(line, end="")

f.close()
```

- We do not read the contents of the file, but instead can use a for loop to iterate over it once we have opened it.
- Since files and folders are directly controlled by Operating Systems, we have to be using the **os module** and use its functions to do some things.
- - ***os.stat("myFile.txt")*** → returns information about the file that is stored on harddisk.
- -***os.remove("myFile.txt")*** → removes the file from the harddisk.
- As we do more file writing and reading, we have to take care of portability, since the way we access files differ greatly on different OS.
- **Examples:** Windows: "Desktop\\photo.jpg" but it is written 'Desktop/photo.jpg' on Mac. These \\ on Windows and / on Mac are generally called path separators, and if you incorrectly use different path separators, you might fail to find the file.
- Hardcoding the file's path takes you nowhere. Instead use the sub module of **os** called **path** on which you can call some functions.
- 1) The first one is called **os.path.join( )** in whose parentheses you pass the single path names without any path separator, and luckily the path separator is generated depending on the type of the **os.** So ***os. path. join("Desktop", "photo.jpg")*** → would generate "Desktop\\photo.jpg" on Windows and "Desktop/photo.jpg" on Mac.
- Sidenote: As a second argument for the **open()** function we can pass the mode that you permit to happen. E.G ***1***→ r means read, so it is only read mode.
- ***READING 18***
- A break statement in Python is used to exit a **loop**.
-