Introduction to Python

UCSAS 2024

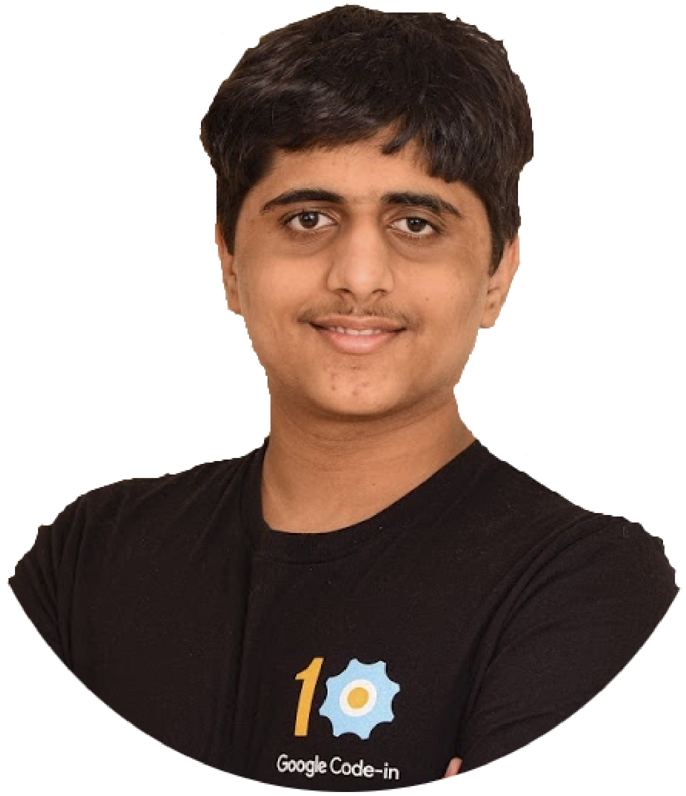
Charitarth Chugh

About Me

- 3rd Year Computer Science Student at UConn
- President of UConn AI Club

Interests

- Deep Learning
- Linux
- Software Development



Why Python?

- Python is a language with very diverse applications, from software development to research
- As a language, it is easy to understand because there is an emphasis on readability.
- The ecosystem of libraries and tools is awesome, which makes finding niche packages a breeze.
    - If you are not able to find anything that suits your needs, it is fairly easy to create a python package of your own as well.
- The only place where Python is potentially not used is when speed is critical.

Prerequisites:

A device with Internet access, preferably a laptop.

What we will be covering today!

- Python Syntax (Variables, Indentation, Comments)
- Data Types and Methods
    - Strings (`str`)
    - Numerical types (`int` , `float` , `complex`)
    - Mapping (`dict`)
    - Sets (`sets` , `frozenset`)
- Conditions, Loops and Functions

Syntax

- To comment a line, prefix it with a `#`
- In Python, a new line indicates a start of a new command

```python
# Print Hello, UCSAS
print("Hello, UCSAS!")

print("Workshop going good?")
```

Numerical and Boolean Data Types

- Integer (`int`)
- Float (`float`)
- complex (`complex`)
  - Ex. `1 +3j` where 3j is the complex component
- Boolean (`bool`)
  - Difference here is true is `True` and false is `False`

Data Types (continued)

Strings

- String is an array of bytes representing Unicode characters and thus elements can be accessed.
- Multiline strings need three quotes and keep line breaks intact while printing.
- Operators like + concatenates strings, in searches for membership
- Commonly used methods:
    - `replace("a", "b")`: replaces a with b in string
    - `split()`: splits based on given separator
    - `upper()`, `lower()`, `strip()`, `capitalize()`, `casefold()`,
    - `count('a')`, `endswith()`, `startswith()`, `find()`, `index()`

Sidenote: types of strings

F-strings:

```
>>> f"{} is 3"

"3 is 3"
```

- Makes it very easy to insert values and expressions into a string.
- Very useful in print statements to see where your code is not working 🙂

R-strings

```
r"./practice.txt"
```

- Ensures that the contents of the strings is treated as-is (no escape characters for example)
- Especially useful when you are working with files

Practice time!

Q1: Manipulating Strings, 4 minutes

Try not to look things up

https://link.charitarth.dev/ucsas-practice

Data Types (continued)

Lists

```
["a", 1, True]
```

- Can contain any type of elements & even a function.
- Indexed and Ordered as a sequence.
- Index starts at 0, like most programming languages
- Elements can be accessed in the following ways: `x[0]`, `x[-1]`, `x[0:3]`, `x[:2]`, `x[2:]`
- Elements can be modified using `x[1:2] = ["UCSAS", [1,2,3]]`
- Methods: `len(x): length`, `x.insert(2, "Python")`, `x.append("Python")`, `x.remove([1, 2])`, `x.pop(1)`, `x.sort()`, `y = x.copy()`, `x.extend(y)`

Data Types (continued)

Tuples

```
(1, 2, [1, 2], 1, "abc")
```

- Created using `()`
- Very similar to a list, but elements inside cannot be changed or be added (immutability)
    - This means that for any change, a new tuple has to be created
- Accessing items is similar to a list.
- Methods: `len(x)`, `x.count("a")`, `x.index()`

Data Types (continued)

Sets

- Like a list, but cannot include duplicate elements
- A set is not able to be indexed, so the only way to access the elements is to loop over it

```
>>> a =  {1, "2"}
```

Data Types (Continued)

Dictionary

- a `dict` is a mapped data type
    - It consists of a key-value pair, where a key is used to access a value.
- The keys of a dictionary are immutable & duplicate keys will replace the original value, but the values themselves are mutable

```
>>> ucsas = "{"workshop": "Introduction To Python", "year": 2022}
>>> ucsas["workshop"]
"Introduction To Python"
>>> ucsas["year"]
2022
>>> ucsas.keys()
["workshop", "year"]
```

Data Types (Continued)

Duck Typing & type enforcement

- Python does not do data type checking.
- It behaves on the principle: "If it walks like a duck and quacks like a duck, then it must be a duck"

Data Types

- Obviously this is not an exhaustive list
- So if you ever need to inspect the type of something, there is a nice built-in `type()` that finds the type.

Variable Assignment

- Is as simple as writing the name of the variable `=` to some value.
- There is no need to define the type of the variable in Python, as it is determined on its own.

```python
# Integer Assignment
x = 2
# String
z = "UCSAS"
## Boolean
w = True
print(x)
print(z)
print(w)
```

Practice time!

Q2: Manipulating lists

5 min

Operators

Arithmetic Operators:

- add: `+`, subtract: `-`, multiply: `*`, division: `/`, modulus: `%`, exponentiation: `**`, floor division: `//`

Assignment Operators:

- equals: `=`, add and equal: `+=`, subtract and equal: `-=`, multiply and equal: `*=`, divide and equal: `/=`

Comparison Operators:

- value equality: `==`, value not equal: `!=`, value greater than: `>`, value less than: `<`, value greater than equal: `>=`, value less than equal: `<=`

Logical Operators:

- `and`, `or`, `not`, `in`

Conditionals (If, elif, else)

- The conditionals should be based on a logical input such as ==, >=, >, <, <=, is, is not, in, not in.
- They can be written in one line if the statement has only one statement.
- An if statement cannot be empty. If it has to be, use pass
- If condition are to result in more than two cases, use elif
  and or can be used for the conditional.
- At the end of the condition to verify, use a : and then if a new line is used, use indentation.

Conditionals (continued)

```python
## checking three scores and using `and`.
a, b, c = 55, 60, 90

if a > b and a > c:
    print('a is first')
elif a < b and b < c:
    print('c is first')
else: print('b is first')


## checking between two scores in one line
a, b = 55, 70

print('a is first') if a > b else print('b is first')
```

Loops

While loop

- It runs as long as a condition is true. Careful as it can run into an infinte loop if condition never gets satisfied.
- `break` and `continue` allows to either break or continue based on a condition within the loop.

```
num = 9380949384209
count = 0
while num!=0:
  if num < 0:
    break
  else:
    continue
  num %= 10
  count += 1
```

Loops (Continued)

For loop

- Used to iterate over a sequence.
- `range()` function is useful as it gives a list of integers to iterate over

```python
x = ['usual', 'usual', 'usual', 'amazing', 'usual', 'usual', 'exit']
count = 0
for i in range(len(x)):
    if x[i] == 'usual':
        count += 1
print(count)

count = 0
for temp in x:
    if temp == 'usual':
        count += 1
print(count)
```

Functions

- A function is defined using keywords def followed by the function name and arguments within parenthesis.
- A function should either print or return some value. Else pass should be used to avoid error.
- Often when we use functions to obtain values and store them in another variable, we need a return statement.
- A lambda is a small anonymous function which returns the result in the same line (a useful property).

```
def fib(n):
    if (n==1 or n==2) return 1
    else return fib(n - 1) + fib(n - 2)
```

Scope

- Scope: A variable created inside a function has only local scope which means it can only be used inside the function. It can be accessed by another function if it is within the previous function.
- A variable created in the main body has global scope.

Practice 3

Catalan Numbers

6 minutes

https://link.charitarth.dev/ucsas-practice

Classes

- A class is a blueprint for objects
- It defines ways to initiate an object of the made up class, functions for various properties, methods, etc.
- **init**(self, parameters) is a function that exists for all classes - to initiate values to the class.
- Methods are defined for the object class using functions with parameter self and more within the class.

```
class Gene:
    def __init__(self, creationid, creationseq):
        self.id = creationid
        self.seq = creationseq

    def length(self):
        return(str(len(self.seq)))

    # search for first instance
    def search(self, searchseq):
        return(self.seq.find(searchseq))

print('Gene Object: Length and search')

x = Gene(11, 'AGTCATCGA# Important References
- W3Schools
- FreeCodeCamp
# AcACTG')
print(f'Gene length:{x.length()}')
print(f"Gene search for AA:{x.search('AA')}")
print(type(x))
```

Sidenote: Getting help in Python

For any object, you can call the `dir()` function to see all the methods that it support

```
>>> dir(list)
['__add__', '__class__', '__class_getitem__', ...]
```

For any function, you can call the `help()` function to read more about what the function does and what its arguments represent

```
>>> help(sorted)
sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

Practice 4

Getting Area

https://link.charitarth.dev/ucsas-practice

Thank you for having me!

Resources:

- W3Schools
- FreeCodeCamp

Acknowledgements

- Thank you to Dr. Yan for letting me do this presentation

https://charitarth.dev

Email me at contact@charitarth.dev if you have any questions!