Basics

Introduction

Python is a high-level, object-oriented and dynamically-typed language with garbage-collection (automatic memory management).

Data Types

str, int, float, list, tuple, dict, set, bool

Variables, types, and values

Operators, expressions, and program output

Strings

Strings in python are like arrays. They can be looped through with a for loop, len() returns the length of the string. But they are immutable, and editing a string will result in a new string.

When printing strings, use fstrings:
print(player_name) -> `print(f'Your name is, {player_name}')`

Lists and Tuples

Lists are a data type that uses consecutive memory in storage to store information. It can be used as a stack, a queue, a tree, or a graph.

Tuples are a data type that consist of a number of values separated by commas. They are immutable but could contain mutable objects, and they may be nested.

If a mutable object in an immutable object is changed, since python uses pointers behind the scenes, the immutable object's contents, in this case the list of the tuple, will update correspondingly. Note that it is still the same list element, no copies were created.

Loops

for i in range(len(string))
for i, n in enumerate(iterable)
while (condition)

File Input and Output

file = open("file name", "Access mode") # r, r+, w, w+, a, a+, b
f.write(str)
f.writelines(str)
f.read(n) # reads the first n contents of the file
f.readline()
file.close()

Iterators

Iterators are objects that iterate over iterable objects. An iterable object is one that can return one of their elements at a time, such as lists, tuples, and strings.

```
while True:
   iterator = iter(iterable)
   element = next(iterator)
   print(element)
```

In python, the StopIteration exception is raised when an iterator runs out of steam. There is no graceful control like the comparisons with the end of iterable as seen in C++. This is because iterators are usually used implicitly in python.

Dictionaries

Dictionaries are python's interpretation of hashmaps. They are exceedingly powerful in so many situations that a common mantra of competitive programming is as follows: when in doubt, dictionary. Its quick, O(1) access and update time, combined with its intuitive and simple structure, makes it one of the most versatile data storage types in all of computer science. Dictionaries can also be nested to create powerful representations of hierarchical data.

```
bank = {'jack': 4098, 'sape': 4139}
print(bank['jack'])
bank['john'] = 600
```

```
del bank['sape']
for name, balance in bank.items():
   print(f"{name} has a balance of {balance}.")
```

Each key must be unique and immutable. The get() method allows a way to access an element with a failsafe that returns a default value if it was not found, which mitigates keyError exceptions.

OOP

The four pillars of OOP are:

- Encapsulation: each object should control its own state. Instead of direct access, use data hiding and setters. Getters should also be used.
- Abstraction: make classes and functions easy to use with detail hiding.
- Inheritance: relatives (children) of an object should acquire the properties and methods of another object. A child class should have all the functionality of its parent class, and if used in its place, it should perform the same. This means that no overridden method should change expected behavior.
- Polymorphism: methods should do different things based on the object it's called on. This is typically achieved through method overriding.

All pillars promote code reuse.

```
class Animal:
   def __init__(self, name):
       self.name = name

   def eat(self):
       return f"{self.name} is eating."

   def sleep(self):
       return f"{self.name} is sleeping."

class Human(Animal):
   def think(self):
       return f"{self.name} is thinking."

animal = Animal("Leo the Lion")
human = Human("Alice")

print(animal.eat())    # Leo the Lion is eating
print(animal.sleep()) # Leo the Lion is sleeping

print(human.eat())     # Alice is eating (inherited)
print(human.sleep())   # Alice is sleeping (inherited)
```

```
print(human.think())  # Alice is thinking (specific to Human)
```

Comprehensions

List comprehensions are more concise ways to create lists. For a loop that performs operations incrementally, for example, we can use list comprehensions:

```
squares = [x ** 2 for x in range(10)]
```

Shallow vs Deep Copies

A shallow copy creates a new object but does not create copies of nested objects, instead it just uses references to those nested objects.

A deep copy creates a new object and recursively copies all the objects it contains.

In python, the assignment operator does not create a copy at all, merely just binding a name to the existing object.
Shallow copies can be created using methods like copy()
Deep copies can be created using methods like deepcopy()