coderss_world

PYTHON CHEAT SHEET 2024

Python Basics

1. Variables & Data Types

```
# Assigning variables
x = 5
name = "John"

# Basic Data Types
integer = 10 # int
floating = 10.5 # float
boolean = True # bool
string = "Hello" # str
```


2. Lists

```
# Creating a list
my_list = [1, 2, 3, "apple", "banana"]
```

```
# Accessing elements
print(my_list[0])  # 1

# Slicing
print(my_list[1:3]) # [2, 3]

# Adding elements
my_list.append("cherry")

# Removing elements
my_list.remove("apple")
```

3. Tuples

```
# Creating a tuple
my_tuple = (1, 2, 3)

# Accessing elements
print(my_tuple[1]) # 2
```

```
# Unpacking tuples
 a, b, c = my_tuple
```

4. Dictionaries

```
# Creating a dictionary
 my_dict = {"name": "John", "age": 25}

 # Accessing values
 print(my_dict["name"]) # John

 # Adding key-value pairs
 my_dict["city"] = "New York"

 # Removing key-value pairs
 del my_dict["age"]
```

5. Sets

1. Defining Functions

```
def greet(name):
  return "Hello, " + name

print(greet("Alice")) # Hello, Alice
```

2. Lambda Functions

```
# Simple lambda
add = lambda x, y: x + y
print(add(2, 3)) # 5

# Sorting with lambda
my_list = [(1, 'one'), (2, 'two'), (3, 'three')]
my_list.sort(key=lambda x: x[1])
```

3. *args and **kwargs

```
def foo(*args, **kwargs):
  print(args)
  print(kwargs)
```

```
foo(1, 2, 3, name="John", age=25)
```

Common Libraries

1. NumPy

```
import numpy as np

 # Creating an array
 arr = np.array([1, 2, 3])

 # Basic operations
 arr = arr * 2 # [2, 4, 6]
```


2. Pandas

```
import pandas as pd

 # Creating a DataFrame
 data = {'Name': ['John', 'Anna'], 'Age': [28, 24]}
 df = pd.DataFrame(data)
```

```
# Creating a set
my_set = {1, 2, 3}

# Adding elements
my_set.add(4)

# Removing elements
my_set.remove(2)
```

Control Flow

1. If Statements

```
x = 10

if x > 5:
print("x is greater than 5")
elif x == 5:
print("x is 5")
else:
print("x is less than 5")
```

2. Loops

```
# For loop
for i in range(5):
  print(i)

# While loop
count = 0
while count < 5:
  print(count)
  count += 1
```

3. List Comprehensions

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

# With a condition
evens = [x for x in range(10) if x % 2 == 0]
```


Functions

```
# Accessing response content
print(response.json())
```

```
# Accessing data
print(df['Name'])

# Basic operations
df['Age'] = df['Age'] + 1
```

3. Matplotlib

```
import matplotlib.pyplot as plt

# Simple line plot
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```


4. Requests

```
import requests

# Making a GET request
response =
data")
requests.get("https://api.example.com/
```

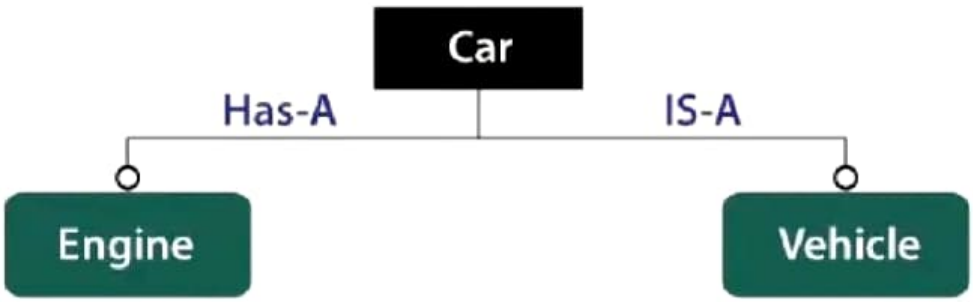
Q 11. What is a final class?

Ans: A final class is a class that cannot be subclassed. It prevents other classes from extending it and inheriting its behavior.

Q 12. What is composition?

Ans: Composition is a design principle where a class contains objects of other classes as part of its attributes.

It allows creating complex structures by combining simpler classes.



Q 13. What is a super keyword?

Ans: The **super keyword** is used to refer to the parent class or superclass. It can be used to call methods and constructors from the superclass.

Vikram
@code._learning

50 Python Interview Q/A

@code._learning

1. What is Python?

   - Python is a high-level, interpreted programming language with dynamic semantics, known for its ease of learning and readability.

2. What are the key features of Python?

   - Python's key features include easy-to-read syntax, dynamic typing, memory management, and a comprehensive standard library.

3. How is memory managed in Python?

   - Memory in Python is managed by the Python memory manager. Objects and data structures are stored in a private heap, and the garbage collector recycles unused memory.

4. What are decorators in Python?

   - Decorators are a design pattern in Python that allows users to modify the behavior of a function or class.

5. What is PEP 8?

   - PEP 8 is the Python Enhancement Proposal that provides guidelines and best practices on how to write Python code.

6. What is a lambda function in Python?

   - A lambda function is a small anonymous function that can take any number of arguments but can only have one expression.

7. What is the difference between list and tuple?

   - The main difference is that lists are mutable while tuples are immutable.

8. How does Python handle the memory deallocation?

   - Python has a built-in garbage collector, which recycles all the unused memory so that it can be made available for heap space.

17. What are Python's generators?

- Generators are a simple way of creating iterators. They return a lazy iterator that can be looped through.

18. What is __init__?

- __init__ is a method or constructor in Python. This method is automatically called to allocate memory when a new object/instance of a class is created.

19. What is self in Python?

- self represents the instance of the class and binds the attributes with the given arguments.

20. What is __str__?

- __str__ is a built-in function in Python that is called when the following functions are invoked on the object: print() or str().

21. What is the difference between append() and extend() methods?

- append() adds its argument as a single element to the end of a list
  while extend() adds each element of its argument to the list.

22. What is a docstring in Python?

- A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.

23. What is the difference between global and local variables?

- Global variables are accessible throughout the program, and local variables are accessible only within the scope of the function where they are declared.

24. What is the pass statement in Python?

- The pass statement is a null operation; nothing happens when it executes.

9. What is slicing in Python?

- Slicing in Python is a mechanism to select a range of items from sequence types like list, tuple, strings etc.

10. What are Python modules?

- Python modules are .py files that consist of Python code. Any Python file can be referenced as a module.

11. What is the difference between Python Arrays and lists?

- Arrays can only contain elements of the same data type, while lists can contain elements of different data types.

12. What is the difference between deepcopy and copy?

- deepcopy creates a new compound object and then, recursively, inserts copies into it of the objects found in the original. copy creates a new compound object and then inserts references into it to the objects found in the original.

13. What is a namespace in Python?

- A namespace is a naming system used to ensure that names are unique to avoid naming conflicts.

14. What is a dictionary in Python?

- A dictionary in Python is an unordered collection of data values used to store data values like a map.

15. What is the difference between xrange and range?

- xrange returns the xrange object while range returns the list, and xrange uses the same memory location.

16. What is pickling and unpickling?

- Pickling is the process whereby a Python object hierarchy is converted into a byte stream, and unpickling is the inverse operation.

25. What is the difference between == and is?

- == checks for equality, while is checks for identity.

26. What is a session in Python?

- A session allows you to persist certain parameters across requests.

27. What is the difference between break, continue, and pass?

- break terminates the loop, continue skips the current iteration, and pass does nothing and acts as a placeholder.

28. What is *args and **kwargs?

- *args is used to pass a variable number of arguments to a function, **kwargs allows you to pass keyworded variable length of arguments to a function.

29. What is the difference between isinstance() and type()?

- isinstance() checks if an object is an instance of a class or a subclass thereof, type() returns the type of the object.

30. What is the difference between .py and .pyc files?

- .py files contain the source code of a program, whereas .pyc files contain the bytecode which can be executed by the Python virtual machine.

31. What is __name__ in Python?

- __name__ is a built-in variable which evaluates to the name of the current module.

32. What are metaclasses in Python?

- Metaclasses are classes of classes that define how a class behaves.

33. What is monkey patching in Python?

- Monkey patching is a technique to add, modify, or suppress the default behavior of a piece of code at runtime.

34. What is the with statement in Python?

- The with statement simplifies exception handling by encapsulating common preparation and cleanup tasks in so-called context managers.

35. What is the difference between staticmethod and classmethod?

- staticmethod does not receive an implicit first argument, while classmethod receives the class as an implicit first argument.

36. What is the difference between .py files and .pyw files?

- .py files are Python source files. .pyw files are Python script files meant to be run on the Windows platform without opening a command prompt window.

37. What is the difference between assert and raise?

- assert is used for debugging purposes while raise is used to raise exceptions.

38. What is the enumerate function in Python?

- enumerate is a built-in function that adds a counter to an iterable and returns it in a form of enumerate object.

39. What is the difference between @staticmethod and @classmethod?

- @staticmethod defines a static method which does not receive an implicit first argument, while @classmethod defines a class method which receives the class as an implicit first argument.

40. What is the difference between __new__ and __init__?

- __new__ is a static method that is called to create an instance, while __init__ is the constructor that is called to initialize the instance.

41. What is the difference between __getattr__ and __getattribute__?

- __getattr__ is called when an attribute lookup has not found the attribute in the usual places, __getattribute__ is called before looking at the actual attributes on the object.

42. What is the global keyword in Python?

- The global keyword is used to declare that a variable inside the function is global (outside the function).

43. What is the difference between __call__ and __init__?

- __call__ allows an instance of a class to be called as a function, __init__ is the constructor method for a class.

44. What is the difference between __dict__ and __dir__?

- __dict__ is a dictionary or other mapping object used to store an object's (writable) attributes, __dir__ is used to list the attributes of the object.

45. What is the super function in Python?

- super is used to give access to methods and properties of a parent or sibling class.

46. What is the difference between __str__ and __repr__?

- __str__ is used for creating output for end user while __repr__ is used for debugging and development. repr is more precise than str.

47. What is the zip function in Python?

- zip is a built-in function that returns an iterator of tuples based on the iterable objects.

48. What are unit tests in Python?

- Unit tests are tests written to check the functionality of a specific section of code, usually at the function level.

49. What is the Global Interpreter Lock (GIL) in Python?

- The GIL is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes at once.

50. What are function annotations in Python?

- Function annotations provide a way of associating various parts of a function with arbitrary python expressions at compile time.

WEB DEVELOPER
@COMPUTER__PROGRAMMER

PYTHON NOTES

Basic Syntax and Structure

- **Indentation**: Python uses indentation to define blocks of code. Consistent use of spaces or tabs is crucial.

- **Variables**: Variables are dynamically typed, meaning you don’t need to declare their type.

- **Comments**: Single-line comments start with **#**, and multi-line comments are enclosed in triple quotes (''' or """).

Data Types

- **Numbers**: Integers, floating-point numbers, and complex numbers.
- **Strings**: Immutable sequences of Unicode characters. Defined using single, double, or triple quotes.
- **Lists**: Ordered, mutable collections of items.
- **Tuples**: Ordered, immutable collections of items.
- **Sets**: Unordered collections of unique items.
- **Dictionaries**: Unordered collections of key-value pairs.

Functions

- **Definition**: Functions are defined using the **def** keyword.
- **Arguments**: Functions can have default, keyword, and variable-length arguments.
- **Return**: Functions use the **return** statement to send back a value.

Modules and Packages

- **Modules**: Files containing Python code (functions, classes, variables).

- **Packages**: Directories containing multiple modules, using an **__init__.py** file to distinguish them.

Object-Oriented Programming (OOP)

- **Classes**: Defined using the **class** keyword. They encapsulate data and functions that operate on data.
- **Objects**: Instances of classes.
- **Inheritance**: Mechanism to create a new class using details of an existing class without modifying it.
- **Polymorphism**: Ability to use a common interface for multiple data types.
- **Encapsulation**: Restricting access to some of an object's components.

Libraries and Frameworks

- **Standard Library**: Python's extensive standard library includes modules for handling OS, file I/O, system administration, protocols, and more.

- **Third-Party Libraries**: Many libraries available via PyPI, such as NumPy for numerical operations, Pandas for data manipulation, and Flask/Django for web development.

Data Science & Machine Learning

- **NumPy**: For numerical operations and managing arrays.
- **Pandas**: For data manipulation and analysis.
- **Matplotlib/Seaborn**: For data visualization.
- **Scikit-learn**: For machine learning algorithms.

WEB DEVELOPER

@COMPUTER__PROGRAMMER

PYTHON CHEATSHEET 2024

1. VARIABLES AND DATA TYPES

```
# Numbers
x = 5   # int
y = 3.14   # float

# Strings
name = "Python"

# Boolean
is_active = True
```

4. TUPLES

```
my_tuple = (1, 2, 3)

# Access elements
first_item = my_tuple[0]

# Unpacking
a, b, c = my_tuple
```

5. DICTIONARIES

```python
my_dict = {"name": "Python", "age": 30}

# Access value by key
name = my_dict["name"]

# Add or update key-value pair
my_dict["version"] = 3.9

# Remove key-value pair
del my_dict["age"]
```

2. STRING OPERATIONS

```
# Concatenation
greeting = "Hello, " + name

# f-strings (formatted strings)
message = f"Hello, {name}!"

# String methods
message.lower()  # Convert to lowercase
message.upper()  # Convert to uppercase
message.split(",")  # Split by a delimiter
```

3. LISTS

```
my_list = [1, 2, 3, 4]

# Access elements
first_item = my_list[0]

# Add item
my_list.append(5)

# Remove item
my_list.remove(3)

# Slicing
sub_list = my_list[1:3]  # Returns [2, 3]
```

- **WHILE LOOP:**

```
count = 0
while count < 5:
    print(count)
    count += 1
```

8. FUNCTIONS

```
def greet(name):
    return f"Hello, {name}!"

greeting = greet("Python")
```

6. CONDITIONALS

```
if x > 5:
    print("x is greater than 5")
elif x == 5:
    print("x is 5")
else:
    print("x is less than 5")
```

7. LOOPS

- **FOR LOOP:**

```
for i in range(5):
    print(i)  # Prints 0 to 4
```

1. BASIC SYNTAX

```
// Single line comment
/* Multi-line comment */

let x = 10; // Variable declaration
const y = 20; // Constant declaration
```

2. DATA TYPES

```
let str = "Hello"; // String
let num = 100; // Number
let bool = true; // Boolean
let arr = [1, 2, 3]; // Array
let obj = { name: "John", age: 25 }; // Object
```

WEB DEVELOPER
@COMPUTER__PROGRAMMER

ALL PYTHON FUNCTIONS

id(): Returns the identity (memory address) of an object.

input(): Reads input from the user as a string.

int(): Converts a value to an integer.

isinstance(): Checks if an object is an instance of a class.

issubclass(): Checks if a class is a subclass of another class.

iter(): Returns an iterator for an iterable object.

len(): Returns the length of an object (e.g., list, string).

list(): Creates a new list.

locals(): Returns a dictionary of the current local symbol table.

map(): Applies a function to every item in an iterable

max(): Returns the largest item in an iterable or two or more arguments.

memoryview(): Returns a memory view object.

min(): Returns the smallest item in an iterable or two or more arguments.

next(): Retrieves the next item from an iterator.

object(): Creates a new featureless object.

oct(): Converts an integer to an octal string.

open(): Opens a file and returns a file object.

ord(): Converts a character to its Unicode integer.

pow(): Returns the value of a number raised to a power.

print(): Prints the given object(s) to the console.

property(): Returns a property attribute.

range(): Returns a sequence of numbers.

repr(): Returns a string representation of an object.

reversed(): Returns a reversed iterator.

round(): Rounds a floating-point number to the nearest integer.

set(): Creates a new set object.

setattr(): Sets the value of a named attribute of an object.

slice(): Returns a slice object.

sorted(): Returns a sorted list of the given iterable.

staticmethod(): Converts a method into a static method.

abs(): Returns the absolute value of a number.

all(): Returns True if all elements in an iterable are true.

any(): Returns True if any element in an iterable is true.

ascii(): Returns a string representation of an object but escapes non-ASCII characters.

bin(): Converts an integer to a binary string.

bool(): Converts a value to a boolean (True or False).

bytearray(): Returns an array of bytes.

bytes(): Returns an immutable byte object.

callable(): Returns True if the object is callable (e.g., a function).

chr(): Converts an integer to a character (based on Unicode).

str(): Converts a value to a string.

sum(): Sums the items of an iterable.

super(): Returns a proxy object that delegates method calls to a parent or sibling class.

tuple(): Creates a new tuple.

type(): Returns the type of an object or creates a new type.

vars(): Returns the __dict__ attribute of an object.

zip(): Combines multiple iterables into tuples.

filter(): Filters elements from an iterable based on a function.

float(): Converts a value to a floating-point number.

format(): Formats a string or value.

frozenset(): Returns an immutable frozenset.

getattr(): Returns the value of a named attribute of an object.

globals(): Returns the current global symbol table as a dictionary.

hasattr(): Returns True if an object has a given attribute.

hash(): Returns the hash value of an object.

help(): Invokes the built-in help system.

hex(): Converts an integer to a hexadecimal string.

classmethod(): Converts a method into a class method.

compile(): Compiles source into a code or AST object.

complex(): Returns a complex number.

delattr(): Deletes an attribute from an object.

dict(): Creates a new dictionary.

dir(): Returns a list of attributes and methods of an object.

divmod(): Returns the quotient and remainder of a division.

enumerate(): Adds a counter to an iterable and returns it.

eval(): Evaluates a Python expression from a string.

exec(): Executes a Python code object or string.

Save & Share

PYRAMIDS IN PYTHON

#1. Pyramid

```
print("\nNormal Pyramid")
for i in range(5):
    x='*'
    x=x*i
    print(f'{x: ^10}')
```

Normal Pyramid

#2. Invert Pyramid

```
print("\nInvert Pyramid\n")
for i in range(5):
    x='*'
    x=x*(5-i)
    print(f'{x: ^10}')
```


Invert Pyramid

#3. Left sided Pyramid

```
print("\nLeft sided Pyramid")
for i in range(5):
    x='*'
    x=x*i
    print(f'{x: <10}')
```

Left sided Pyramid

#4. Right sided Pyramid

```
print("\nRight sided Pyramid")
for i in range(5):
    x='*'
    x=x*1
    print(f'{x: >10}')
```

Right sided Pyramid