PYTHON CHEATSHEETS

Print Hello World

```
print("Hello World!!")
```

Printing Output

```
name = "Rahul"

print("My name is: ", var1)
```

Taking Input From User

```
let  = input("Enter your Name: ")

print("Hi my name is: ", name)
```

Python Comments

It is used to make the code more readable.

Single-Line Comments

Single-line comments start with the hash symbol (#).

```
#This is a single line comment
print("Hello World!!")
```

Multi-Line Comments

To write multi-line comments you can use (#) at each line.

```
#This is a
#multi line
#comment
print("Hello World!!")
```

Escape Sequence

Newline

Newline Character

```
print("\n")
```

Backslash

It adds a backslash

```
print("\\")
```

Single Quote

It adds a single quotation mark

```
print("\'")
```

Variables

Variables are used to store data values. Python is a dynamically typed language, which means you don't need to declare a variable's type directly.

Example:

```
name = "john"  #type str
age = 22       #type int
```

Data Types

Data types in Python represent the types of values that a variable can hold. Python supports various built-in data types, including:

Numeric Types

- **int:** 4, -6, 0
- **float:** 3.14, 2.0.

- **complex** 5 + 3i

Text Type

- **str:** “Hello Python”

Boolean Type

- Boolean data consists of True or False values.

Sequence Types

- List
- Tuple
- Set
- Dict

Numbers

In Python, numerical data types are classified into three types:

- int
- float
- complex

int

Integers are whole numbers, either positive or negative, with no decimal points.

Example:

```
x = 24
y = 100548

print(type(x))
print(type(y))
```

float

Floating-point numbers are real numbers with a decimal point.

Example:

```
x = 2.34
y = 3.1

print(type(x))
print(type(y))
```

complex

Complex numbers are made up of both real and imaginary numbers.

Example:

```
x = 6j
y = -6j

print(type(x))
print(type(y))
```

Operators

Arithmetic operators

Python arithmetic operators are + (addition), - (subtraction), * (multiplication), / (division), // (floor division), % (modulus).

Assignment operators

Python assignment operators are used to assign values to variables, including =, +=, -=, *=, /=, %=, **=, //=, &=, |=.

Comparison operators

Python comparison operators are used to compare two values, and include == (equal), != (not equal), > (greater than), < (less than), >= (greater than or equal to), and <= (less than or equal to).

Logical operators

Python logical operators are **and**, **or**, and **not**.

Strings

Python strings are a sequence of characters that are enclosed by double quotes ("") or single quotes (' '). For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', 'o'.

Example:

```
# Double Quotes
str1 = "Hello Python"

# Single quotes
str2 = 'Hello Python'
```

Access String Characters

Example:

```
str1 = "Python"
print(str1[2])
```

Compare Two Strings

We use the == operator to compare two strings.

Example:

```
str1 = "Hello Python"
str2 = "I love Python"
str3 = "Hello Python"

print(str1 == str2)
print(str1 == str3)
```

String Concatenation

To concatenate, two or more strings you can use the + operator.

Example:

```
str1 = "Hello"
str2 = " World"

result = str1 + str2
print(result)
```

String Length

To find the length of a string, use the `len()` function.

Example:

```
str1 = "Hello Python"
print(len(str1))
```

upper() method

The upper() method converts a string to upper case.

Example:

```
str1 = "Hello Python"
print(str1.upper())
```

lower() method

The `lower()` method converts a string to upper case.

Example:

```
str1 = "Hello Python"
print(str1.lower())
```

replace method

The `replace()` method replaces a string with another string.

Example:

```
str1 = "Hello Python"
print(str1.replace("Python", "World"))
```

Lists

A list is an ordered collection of data elements separated by a comma and enclosed within square brackets. They store multiple items in a single variable.

Example:

```
list1 = [20, 40, 60]
print(list1)
```

Access List Items

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
print(flowers[1])
```

append() method

To add an item to the end of the list, use the `append()` method.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]

flowers.append("Blossom")

print(flowers)
```

insert() method

To insert a list item at a specific index, use the `insert()` method.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]

flowers.insert(2, "Blossom")

print(flowers)
```

extend() method

The `extend()` method adds an entire list to the existing list.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers2 = ["Blossom", "Tulip", "Jasmine"]

flowers.extend(flowers2)
```

pop() method

The `pop()` method removes the last item from the list if no index is specified. If an index is provided, the item at that specific index is removed.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers.pop()
print(flowers)
print(flowers)
```

clear()

The `clear()` method clears all items in the list and prints an empty list.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers.clear()
print(flowers)
```

Tuples

A tuple is an ordered collection of data elements separated by a comma and enclosed within parentheses. They store multiple items in a single variable. Tuples are unchangeable meaning we can not change them after creation.

Example:

```
colors = ("Red", "Blue", "White")
print(colors)
```

count() method

The `count()` method returns the number of times the specified items appears in the tuple.

Example:

```
colors = ("Red", "Blue", "White")
newtup = colors.count("White")
print(newtup)
```

index()

The `index()` method returns the index of the first occurrence of the tuple item.

Example:

```
colors = ("Red", "Blue", "White")
newtup = colors.index("White")
print(newtup)
```

Sets

Sets are unordered collection of data items. They store multiple items in a single variable. Sets items are separated by commas and enclosed within curly braces {}.

Add set Items

To add a single item to a set use the `add()` method.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits.add("Banana")
print(fruits)
```

Update Set

To add items from another set into the existing set, use the update() method.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits2 = {"Lemon", "Strawberry", "Litchi"}

fruits.update(fruits2)

print(fruits)
```

isdisjoint() method

The `isdisjoint()` method checks if items of given set are present in another set.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits2 = {"Apple", "Orange", "Mango"}
print(fruits.isdisjoint(fruits2))
```

issuperset()

The `issuperset()` method checks if all the items of a specified set are present in the original set.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits2 = {"Apple", "Mango"}
print(fruits.issuperset(fruits2))
```

Dictionaries

Dictionaries are ordered collection of data items. Dictionaries items are key-value pairs that are separated by commas and enclosed within curly brackets {}.

pop()

The `pop()` method removes the item with the provided key name.

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
details.pop("canVote")
print(details)
```

clear()

The `clear()` method removes all the items from the dictionary.

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
details.clear()
print(details)
```

popitem()

The `popitem()` method removes the last item from the dictionary.

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
details.popitem()
print(details)
```

If Statement

The if statement is used to execute a block of code if a given condition is true.

```
if condition:
  # block of code to be executed if the condition is true
```

If...else statement

The If...else statement is used to execute a block of code if a specified condition is true and another block of code if the condition is false.

```
if condition:
  # block of code to be executed if the condition is true
else:
  # block of code to be executed if the condition is false
```

if…elif…else Statement

Python's if-elif-else statement executes a block of code among multiple possibilities.

```
if (condition1):
  # block of code to be executed if condition1 is true
elif (condition2):
  # block of code to be executed if the condition1 is false and
condition2 is true
else:
  # block of code to be executed if the condition1 is false and
condition2 is false
```

Example:

```
number = 10
if (number > 15):
   print("Number is greater than 15")
elif (number > 10):
   print("Number is greater than 10 but less than or equal to 15")
else:
   print("Number is equal to 10")
```

Loops

A loop or iteration statement repeatedly executes a statement, until the controlling expression is false (0).

for Loop

A for loop in Python is used to iterate over a sequence (e.g., a list, tuple, or string) or any other iterable object.

Example:

```
companies = ["Google", "Facebook", "Microsoft"]
for i in companies:
    print(i)
```

while Loop

While loops in Python are used to execute a block of code several times as long as a condition is true.

Example:

```
number = 1
while (number <= 5):
    print(number)
    number = number + 1
```

Functions

A function is a block of code that executes a specific task when called. They are defined with the def keyword followed by the function name, parentheses (), and a colon.

Example:

```
def my_func():
    print("Hello World")
```

Call a function

To call a function, use the function name followed by parenthesis

Example:

```
def my_func():
    print("Hello World")
my_func()
```

Function Arguments

Arguments are the inputs passed to the function.

Example:

```
def my_func(fname, lname):
    print("Hello", fname, lname)
my_func("John", "Doe")
```

Recursion

Recursion is a programming method that involves calling a function itself to solve a problem.

Example:

```
def fibonacci(n):
    if n == 1 or n == 2:
      return 1
    else:
      return fibonacci(n - 1) + fibonacci(n - 2)
print(fibonacci(10))
```

OOPS

OOPS stand for Object Oriented Programming System. It is a programming paradigm that uses objects and classes in programming.

Class

A class is a blueprint for creating objects. It can be defined using the class keyword, followed by the class name and a colon.

Example:

```
class Student:
    name = "Arka"
    age = 22
```

Objects

An object is an instance of a class.

Example:

```
class Student:
    name = "Arka"
    age = 22

obj1 = Student()
print(obj1.name)
```