Python Cheatsheet

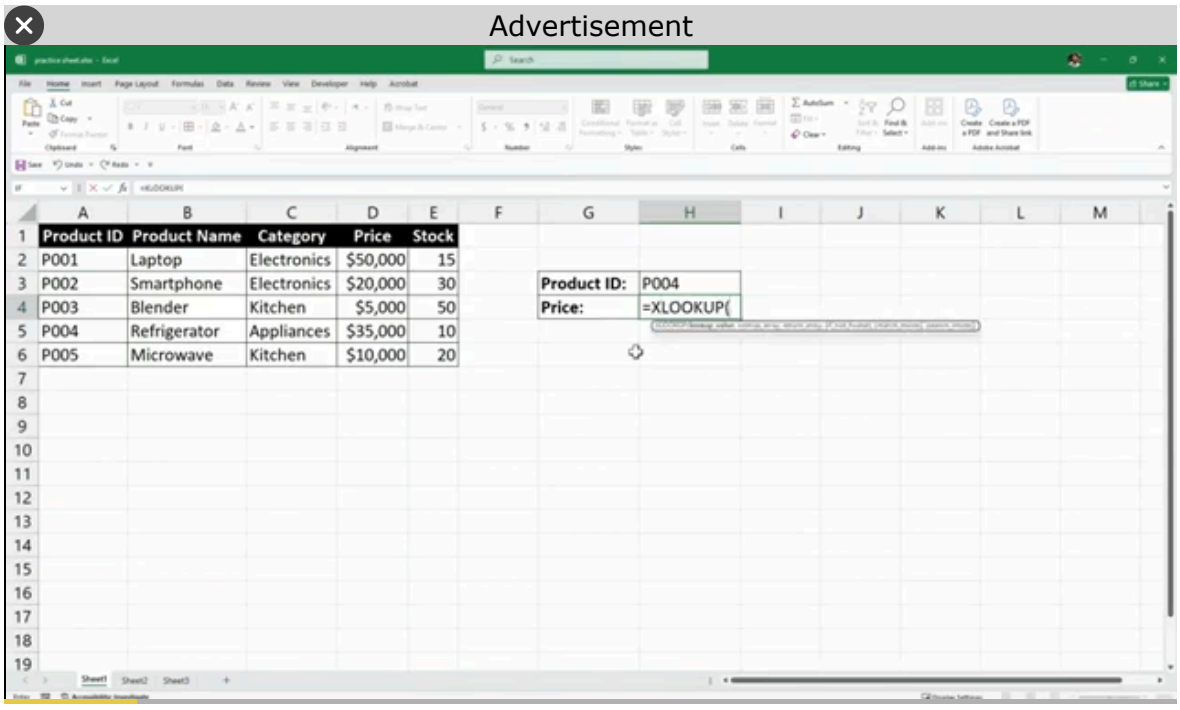
This Python cheatsheet is helpful for students and developers as well to learn Python programming quickly.

Printing Hello World

Printing "Hello, World!" is a basic program. To print "Hello, World," use the print() function. Here is the basic Python program −

```
print("Hello, World!")  # Output: Hello, World!
```

Advertisement



Printing Output

The **print() function** can also be used to print the output, i.e., the value of the variables on the screen.

```
name = "Kelly Hu"
print("Hi my name is: ", name)
```

To print multiple variables, you can use the following cascading form of the print() function –

```
name = "Kelly Hu"
age = 27
city = "Brentwood"
print("I am", name, ". I'm", age, "years old and lives in", city)
```

Print Without Newline

The print() method inserts a newline after printing the result. To print without a newline, use the end=' ' parameter as the last parameter in print() method.

```
print("Hello", end=" ")
print("World")
```

You can specify any value to **end** parameter; the new assigned value will be printed after the result.

```
print("Hello", end="####")
print("World")
```

Comments

Python provides three types of comments –

- Single-line comment
- Multi-line comment
- Docstring comment

1. Single-line comment

Single-line comment is placed followed by the hash (#) single.

```
# This is single-line comment
```

2. Multi-line comment

Multi-line comment is written between the set of three single quotes ('''). Anything is written between ''' and ''' refers to a multi-line comment.

```
'''
name = "Kelly Hu"
age = 30
city = "Brentwood"
'''

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

Variables

Python variables are created when you assign values to them. There is no keyword to declare a variable in Python.

```
name = "Kelly Hu"
age = 27
print(name)
print(age)
```

Specifying Variable's Data Type

The data type of a variable can be specified using the type casting with the help of built-in functions such as str(), int(), float(), etc.

```
a = str(123)
b = int(123)
c = float(123)
# Printing type
print(type(a))
print(type(b))
print(type(c))
```

Printing Type of Variable/Object

The type() function can be used to print the data type (object type) of a variable or an object.

```
name = "Kelly Hu"
print(name, "is the type of", type(name))
```

Variable Names

Variable names (or identifiers) must be started with a letter or the underscore character, and only alpha-numeric characters and underscores are allowed as the variable names.

Some of the valid variable names are: **name**, **_name**, **my_name**, **name2**, **MY_NAME**

Assign Multiple Values to Variables

You can assign multiple values to variables by separating the variable names and values with commas.

```
```

```
name, age, city = "Kelly Hu", 27, "Brentwood"
print(name)
print(age)
print(city)
```

Data Types

The following are the Python data types –

1. Text Types: String (str)

```
text = "Hello, World!"
```

2. Numeric Types

int

```
num_int = 10
```

float

```
num_float = 10.5
```

complex

```
num_complex = 2 + 3j
```

3. Sequence Types

list

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

tuple

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

range

```
my_range = range(1, 10)
```

4. Mapping Type: Dictionary (dict)

```
my_dict = {"name": "Kelly Hu", "age": 27}
```

5. Set Types

set

```
my_set = {1, 2, 3}
```

frozenset

```
my_frozenset = frozenset([1, 2, 3])
```

6. Boolean Type: bool

```
my_bool = True
```

7. Binary Types

bytes

```
my_bytes = b"Hello"
```

bytearray

```
my_bytearray = bytearray(5)
```

memoryview

```
my_memoryview = memoryview(b"Hello")
```

8. None Type: NoneType

```
my_none = None
```

Operators

Python operators are used to perform various operations such as arithmetic, logical, etc. on operands.

Arithmetic operators

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

Assignment operators

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

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**.

Identity operators

Python identity operators are **is** and **is not**, used to check if two variables refer to the same object in memory.

Membership operators

Python membership operators are **in** and **not in**. These operators are used to test whether a value is found in a sequence or not.

Bitwise operators

Python bitwise operators perform operations on binary values; these operators are AND (&), OR (|), NOT (~), XOR (^), left shift (<<), and right shift (>>).

User Input

Python allows you to take different types of input from the user. The input() function is used to take user input as a string from the user. You can use the following basic methods to convert input to the specific type −

- int() − Converts input to an integer.
- float() − Converts input to a float.
- bool() − Converts input to a Boolean.

Basic User Input

```
name = input("Enter your name: ")
print("Hello", name)
```

Integer Input

```
age = int(input("Enter your age: "))
print(f"You are {age} years old.")
```

Float Input

```
x = float(input("Enter any float value: "))
print(x)
```

Handling Invalid Inputs

You can use the try and except to handle the invalid inputs.

```
try:
   x = int(input("Input an integer value: "))
   print(x)
except ValueError:
   print("Please enter a valid number.")
```

Conditions

Python conditional statements are written using the **if**, **elif**, and **else** keywords.

```
a = 10
b = 20
if a > b:
   print("a is greater than b")
elif b > a:
   print("b is greater than a")
else:
   print("a is greater than b")
```

Loops

There are two types of Python loops – the for loop and the while loop.

The for Loop

```
students = ["Kelly Hu", "Akanksha", "Peter"]
for s in students:
   print(s)
```

The while Loop

```
i = 1
while i <= 10:
   print(i)
   i += 1
```

Strings

Python strings are the sequence of characters enclosed within single or double quotation marks.

Printing Strings

```
print("I'm Kelly Hu I live in Brentwood")
name, city = "Kelly Hu", "Brentwood"
# Printing string variables
print("I'm", name, "I live in", city)
```

Quotes Inside Quotes

You can print quotes as a string using the print() function.

```
print("I'm Kelly Hu")
print("I am 'Kelly Hu'")
print("I am \"Kelly Hu\"")
```

Multiline Strings

Python allows to assign multiline strings to a variable. Just place the multiline string using three single or doble quotes.

```
a = """I am Kelly Hu
I lives in Brentwood.
I am 27 years old"""

b = '''I am Kelly Hu
I lives in Brentwood.
I am 27 years old'''

print(a)
print(b)
```

Slicing Strings

String slicing can be done by using the start and the end index, separated by a colon inside square brackets.

```
test = "Hello, World!"
print(test[2:5])
print(test[0:])
print(test[:13])
print(test[:5])
```

String Concatenation

You can concatenate two or more strings using the plus (+) operator.

```
name = "Kelly Hu"
city = "Brentwood"

person = name + " " + city
print(person)
```

String Format

String formatting is done using the Python f-string.

```
name = "Kelly Hu"
age = 27
person = f"My name is {name}, I am {age} years old"
print(person)
```

String Methods

The following are the popular string methods –

Sr.No.	Method & Description
1	**capitalize()** Capitalizes first letter of string
2	**casefold()** Converts all uppercase letters in string to lowercase. Similar to lower(), but works on UNICODE characters alos
3	**lower()** Converts all uppercase letters in string to lowercase.
4	**swapcase()** Inverts case for all letters in string.
5	**title()** Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
6	**upper()** Converts lowercase letters in string to uppercase.

Escape Characters

Python escape characters are the combination of escape (\) and a character that are used for performing specific tasks. The following are the escape characters with their meanings –

Escape Character	Description	Example
\\	Backslash	print("This is a backslash: \\")
\'	Single quote	print('It\'s a sunny day!')
\"	Double quote	print("He said, \"Hello!\"")
\n	Newline	print("Hello\nWorld")
\r	Carriage return	print("Hello\rWorld")
\t	Horizontal tab	print("Hello\tWorld")
\b	Backspace	print("Hello\bWorld")
\f	Form feed	print("Hello\fWorld")
\v	Vertical tab	print("Hello\vWorld")
\ooo	Octal value (character represented by octal number)	print("\101") # Prints 'A'
\xhh	Hexadecimal value (character represented by hex number)	print("\x41") # Prints 'A'

Lists

Python lists are used to store comma-separated values.

Creating Lists

```
# Empty list
my_list = []
# List with elements
my_list = [1, 2, 3, 4, 5]
```

Accessing Elements

Accessing elements of a list can be done using the index and list slicing.

```
my_list = [1, 2, 3, 4, 5]
print(my_list[0])
print(my_list[2])
print(my_list[-1])
print(my_list[2:4])
```

Appending and Inserting Elements

The elements in a list can be appended and inserted using the list.append() and list.insert() methods, respectively.

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.append(6)
my_list.insert(0, 0)
print(my_list)
```

Removing List Elements

The remove() method is used to remove the elements of a list.

```
my_list = [1, 2, 3, 4, 5]
print(my_list)
my_list.remove(3)
print(my_list)
```

Copying Lists

The list.copy() method is used to copy a list to another.

```
```

```
my_list = [1, 2, 3, 4, 5]
new_list = my_list.copy()
print(my_list)
print(new_list)
```

Clearing a List

The list.clear() method is used to clear list elements.

```
my_list.clear()
```

Nested Lists

Python nested list refers to the list within a list.

```
nested_list = [[1, 2], [3, 4], [5, 6]]
print(nested_list)
```

Tuples

Python tuples are used to store multiple comma-separated values in a variable. Tuple values are given inside the round brackets.

Creating Tuples

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

Creating Empty Tuples

```
```

```
my_tuple = ()
print(my_tuple)
```

Tuple Without Parentheses (Implicit Tuple)

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

Single Element Tuple

```
my_tuple = (1,)
print(my_tuple)
```

Accessing Tuple Elements

Tuple elements can be accessed using the index and tuple slicing.

```
my_tuple = (10, 20, 30)
print(my_tuple[0])
print(my_tuple[1])
print(my_tuple[0:])
print(my_tuple[-1])
print(my_tuple[0:2])
```

Tuple Concatenation

To concatenate tuple, use the plus (+) operator.

```
tuple1 = (10, 20, 30)
tuple2 = (40, 50)
print(tuple1 + tuple2)
```

Tuple Unpacking

```
tuple1 = (10, 20, 30)
a, b, c = tuple1
print(a)
print(b)
print(c)
```

Nested Tuples

```
nested_tuple = ((10, 20), (30, 40), (50, 60))
print(nested_tuple)
```

Sets

Python sets are the collection of multiple items that are unordered, unchangeable*, and unindexed. Set values are comma-separated and enclosed within the curly braces.

Creating Sets

```
set1 = {10, 20, 30, 40, 50}
# Set from a list
set2 = set([10, 20, 30, 40, 50])
print(set1)
print(set2)
```

Adding and Removing Elements

The following functions are used to add and remove elements from a set –

- add() – To add a single element.
- update() – To add multiple elements.
- remove() – To remove an element.
- discard() – To remove an element safely.
- pop() – To remove and return an arbitrary element.
- set.clear() – To clear a set.

Example of Set Operations

```
set1 = {10, 20, 30, 40, 50}
set1.add(60)
print(set1)
set1.update([70, 80])
print(set1)
set1.remove(30)
print(set1)
set1.discard(10)
print(set1)
print(set1.pop(),"is removed!")
set1.clear()
print(set1)
```

Dictionaries

Python dictionaries are the collection of key and value pairs and are written with curly brackets.

Creating Dictionary

```
my_dict = {'name': 'Kelly Hu', 'age': 27}
print(my_dict)
```

Creating Empty Dictionary

```
my_dict = {}
print(my_dict)
```

Creating Dictionary with Mixed Keys

```
my_dict = {1: 'Kelly', 'hair_color': 'Brown', (36, 24): 'lucky_numbers'}
print(my_dict)
```

Accessing Values

You can directly access the values of the specified keys by providing the key names inside the square brackets enclosed with quotes, and you can also use the dict.get() method to get the value of the specified key.

```
person = {'name': 'Kelly', 'age': 27, 'city': 'Brentwood'}
# Direct access
print(person['name'])
print(person['age'])
print(person['city'])
# Using method
print(person.get('name'))
print(person.get('age'))
print(person.get('city'))
```

Modifying Dictionaries

You can directly update a value of the existing key, and you can also update the multiple values using the dict.update() method.

```
person = {'name': 'Kelly', 'age': 27, 'city': 'Brentwood'}
print(person)
# Updating single value
person['age'] = 18
print(person)
# Updating multiple values
person.update({'age': 21, 'city': 'New York'})
print(person)
```

Removing Elements

The following methods can be used to remove elements from a dictionary −

- pop() − It removes a key-value pair and return the value. removed_value = person.pop('age')
- popitem(): It removes the last added item and return a key-value pair. removed_value = person.popitem()
- dict.clear(): It remove all items and clears the dictionary. person.clear()

Iterating Through a Dictionary

```
person = {'name': 'Kelly', 'age': 27, 'city': 'Brentwood'}
# Iterate through keys
for key in person:
   print(key)

# Iterate through values
for value in person.values():
   print(value)

# Iterate through key-value pairs
```

```
for key, value in person.items():
   print(f"{key}: {value}")
```

Dictionary Comprehensions

```
dict1 = {x: x**2 for x in range(5)}
print(dict1)
```

Lambdas

Python lambda function can have only one expression. These are small anonymous functions that can take any number of arguments.

```
value = lambda x : x * 3
print(value(5))
```

Classes and Objects

Python is an object-oriented programming language and allows to create classes and objects. You can create a class in Python using the "class" keyword.

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

   def display_info(self):
      print(f"Name: {self.name}, Age: {self.age}")

# Create objects
per1 = Person("Kelly Hu", 27)
```

```
per2 = Person("Adele", 25)

# Access object attributes
print(per1.name)
print(per1.age)

# Printing using method
per2.display_info()
```

Inheritance

Python inheritance allows you to inherit the properties of the parent class to the child class.

Basic Inheritance Example

```
# Base class
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def get_details(self):
        return f"Student: {self.name}, Grade: {self.grade}"

# Derived class
class Graduate(Student):
    def get_details(self):
        return f"Graduate: {self.name}, Grade: {self.grade}, Status: Graduated"

# Create an instance of the child class
grad = Graduate("Kelly Hu", "B+")

# Access inherited and overridden methods
print(grad.get_details())
```

super() with Method Overriding

The super() method is used to call a method from the parent (or base) class.

```
# Base class
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

    def get_details(self):
        return f"Student: {self.name}, Grade: {self.grade}"

# Derived class
class Graduate(Student):
    def __init__(self, name, grade, graduation_year):
        # Call the parent class's constructor using super()
        super().__init__(name, grade)
        self.graduation_year = graduation_year

    def get_details(self):
        # Call the parent class's get_details() method using super()
        student_details = super().get_details()
        return f"{student_details}, Graduation Year: {self.graduation_year}"

# Create an instance of the child class (Graduate)
grad = Graduate("Kelly Hu", "B+", 2011)

# Access inherited and overridden methods
print(grad.get_details())
```

Exception Handling

Python exception handling allows to handle exceptions during execution of the program. The follow keywords (blocks) are used with exception handling –

- **try** – Write the main code on which you want to test the exception.
- **except** – Write code to handle the exception.
- **else** – Write code to execute when there is no error.
- **finally** – Write code to execute finally no matter there is an error or not.

```python
# Create function to divide two numbers
def divide_numbers(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
    else:
        print(f"The divide result is : {result}")
    finally:
        print("Execution complete.")

# Calling function
print("Test Case 1:")
divide_numbers(10, 2)

print("\nTest Case 2:")
divide_numbers(10, 0)
```


TOP TUTORIALS

Python Tutorial

Java Tutorial

C++ Tutorial

C Programming Tutorial

C# Tutorial

PHP Tutorial

R Tutorial

HTML Tutorial

CSS Tutorial

JavaScript Tutorial

SQL Tutorial

TRENDING TECHNOLOGIES

Cloud Computing Tutorial

Amazon Web Services Tutorial

Microsoft Azure Tutorial

Git Tutorial

Ethical Hacking Tutorial

Docker Tutorial

Kubernetes Tutorial

DSA Tutorial

Spring Boot Tutorial

SDLC Tutorial

Unix Tutorial

CERTIFICATIONS

Business Analytics Certification

Java & Spring Boot Advanced Certification

Data Science Advanced Certification

Cloud Computing And DevOps

Advanced Certification In Business Analytics

Artificial Intelligence And Machine Learning

DevOps Certification

Game Development Certification

Front-End Developer Certification

AWS Certification Training

Python Programming Certification

COMPILERS & EDITORS

Online Java Compiler

Online Python Compiler

Online Go Compiler

Online C Compiler

Online C++ Compiler

Online C# Compiler

Online PHP Compiler

Online MATLAB Compiler

Online Bash Terminal

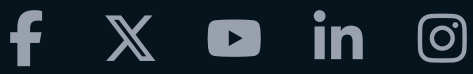
Online SQL Compiler

Online Html Editor

ABOUT US | OUR TEAM | CAREERS | JOBS | CONTACT US | TERMS OF USE |

PRIVACY POLICY | REFUND POLICY | COOKIES POLICY | FAQ'S

tutorialspoint



GET IT ON Google Play

Download on the App Store

Tutorials Point is a leading Ed Tech company striving to provide the best learning material on technical and non-technical subjects.

© Copyright 2025. All Rights Reserved.