Learn Python with me : Beginners to Advanced Level (Hands on Practice)

@zahid007

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

```
Hello World!
```

```
import sys
print(sys.version)
```

```
3.12.11 (main, Jun  4 2025, 08:56:18) [GCC 11.4.0]
```

```
a = 5
b = 7
c = 9
x = a
y = b
z = c
print(x)
print(y)
print(z)

sum = a + b + c
sum1 = x + y + z

print(sum)
print(sum1)

a += 1
b -= 1
c *= 2
x /= 2
y %= 2
z **= 2

print(a)
print(b)
print(c)
print(x)
print(y)
print(z)
```

```
5
7
9
21
21
6
6
18
2.5
1
81
```

```
a = "Wel Come"
b = "Pythom"
c = a + " " +b
print(c)

t = a * 5
print(t)
```

```
print(a,b)
print(a+b)
```

```
Wel Come Pythom
Wel ComeWel ComeWel ComeWel ComeWel Come
Wel Come Pythom
Wel ComePythom
```

Start coding or generate with AI.

Casting

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0

print(x)
print(y)
print(z)

x = 5
y = "John"
z = 1 + 2j
print(type(x))
print(type(y))
print(type(z))
```

```
3
3
3.0
<class 'int'>
<class 'str'>
<class 'complex'>
```

Single or Double Quotes?

String variables can be declared either by using single or double quotes:

```
x = "John"
# is the same as
x = 'John'
print(x)
```

```
John
```

Case-Sensitive

Variable names are case-sensitive.

```
a = 4
A = "Sally"
#A will not overwrite a
```

Start coding or generate with AI.

Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

`**Rules for Python variables:**` **bold text**

-A variable name must start with a letter or the underscore character

-A variable name cannot start with a number

-A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

-Variable names are case-sensitive (age, Age and AGE are three different variables)

-A variable name cannot be any of the Python keywords.

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Multiple Variables

Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)

x = y = z = "Orange"
print(x)
print(y)
print(z)

fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
Orange
Orange
Orange
apple
banana
cherry
```

```
x = "Python is awesome"
print(x)

x = "Python"
y = "is"
z = "awesome"
print(x, y, z)

x = "Python "
y = "is "
z = "awesome"
```

```
print(x + y + z)

x = 5
y = 10
print(x + y)

x = 5
y = "John"
print(x + y)

x = 5
y = "John"
print(x, y)
```

```
Python is awesome
Python is awesome
Python is awesome
15
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-650293994.py in <cell line: 0>()
     18 x = 5
     19 y = "John"
---> 20 print(x + y)
     21
     22 x = 5

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Next steps: Explain error

Global Variables

Variables that are created outside of a function (as in all of the examples in the previous pages) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()

x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)

def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)


x = "awesome"
```

```
def myfunc():
  global x
  x = "fantastic"

myfunc()

print("Python is " + x)
```

```
Python is awesome
Python is fantastic
Python is awesome
Python is fantastic
Python is fantastic
```

Python Data Types

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

None Type: NoneType

```
x = "Hello World"   #str
x = 20  #int
x = 20.5    #float
x = 1j  #complex
x = ["apple", "banana", "cherry"]   #list
x = ("apple", "banana", "cherry")   #tuple
x = range(6)    #range
x = {"name" : "John", "age" : 36}   #dict
x = {"apple", "banana", "cherry"}   #set
x = frozenset({"apple", "banana", "cherry"})    #frozenset
x = True    #bool
x = b"Hello"    #bytes
x = bytearray(5)    #bytearray
x = memoryview(bytes(5))    memoryview
x = None
```

Python Numbers

There are three numeric types in Python:

int

float

complex

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

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

```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))

x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))

x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))

x = 3+5j
y = 5j
z = -5j

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

```
<class 'int'>
<class 'float'>
<class 'complex'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)
```

```
print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Random Number

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

```
import random

print(random.randrange(1, 10))
```

```
7
```

Specify a Variable Type

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3

x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2

x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

Start coding or generate with AI.

Assign String to a Variable

Multiline Strings You can assign a multiline string to a variable by using three quotes:

```
a = "Hello"
print(a)

a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)

a = "Hello, World!"
```

```
print(a[1])

for x in "banana":
  print(x)


a = "Hello, World!"
print(len(a))

txt = "The best things in life are free!"
print("free" in txt)


txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")


txt = "The best things in life are free!"
print("expensive" not in txt)


txt = "The best things in life are free!"
if "expensive" not in txt:
  print("No, 'expensive' is NOT present.")
```

```
Hello
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
e
b
a
n
a
n
a
13
True
Yes, 'free' is present.
True
No, 'expensive' is NOT present.
```

```
b = "Hello, World!"
print(b[2:5])

b = "Hello, World!"
print(b[:5])

b = "Hello, World!"
print(b[2:])

b = "Hello, World!"
print(b[-5:-2])

a = "Hello, World!"
print(a.upper())

a = "Hello, World!"
print(a.lower())

a = " Hello, World! "
print(a.strip()) # returns "Hello, World!"
```

```
a = "Hello, World!"
print(a.replace("H", "J"))

a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

```
llo
Hello
llo, World!
orl
HELLO, WORLD!
hello, world!
Hello, World!
Jello, World!
['Hello', ' World!']
```

```
a = "Hello"
b = "World"
c = a + b
print(c)

a = "Hello"
b = "World"
c = a + " " + b
print(c)

'''age = 36
#This will produce an error:
txt = "My name is John, I am " + age
print(txt)'''

age = 36
txt = f"My name is John, I am {age}"
print(txt)

price = 59
txt = f"The price is {price} dollars"
print(txt)

price = 59
txt = f"The price is {price:.2f} dollars"
print(txt)

txt = f"The price is {20 * 59} dollars"
print(txt)
```

```
HelloWorld
Hello World
My name is John, I am 36
The price is 59 dollars
The price is 59.00 dollars
The price is 1180 dollars
```

Start coding or generate with AI.

Escape Characters

' Single Quote

\ Backslash

\n New Line

\r Carriage Return

\t Tab

\b Backspace

\f Form Feed

\ooo Octal value \xhh Hex value

```
txt = "We are the so-called \"Vikings\" from the north."
```

Start coding or generate with AI.

String Methods

capitalize() Converts the first character to upper case

casefold() Converts string into lower case

center() Returns a centered string

count() Returns the number of times a specified value occurs in a string

encode() Returns an encoded version of the string

endswith() Returns true if the string ends with the specified value expandtabs() Sets the tab size of the string

find() Searches the string for a specified value and returns the position of where it was found

format() Formats specified values in a string

format_map() Formats specified values in a string

index() Searches the string for a specified value and returns the position of where it was found

isalnum() Returns True if all characters in the string are alphanumeric

isalpha() Returns True if all characters in the string are in the alphabet

isascii() Returns True if all characters in the string are ascii characters

isdecimal() Returns True if all characters in the string are decimals

isdigit() Returns True if all characters in the string are digits

isidentifier() Returns True if the string is an identifier

islower() Returns True if all characters in the string are lower case

isnumeric() Returns True if all characters in the string are numeric

isprintable() Returns True if all characters in the string are printable

isspace() Returns True if all characters in the string are whitespaces

istitle() Returns True if the string follows the rules of a title

isupper() Returns True if all characters in the string are upper case

join() Joins the elements of an iterable to the end of the string

ljust() Returns a left justified version of the string

lower() Converts a string into lower case

lstrip() Returns a left trim version of the string

maketrans() Returns a translation table to be used in translations

partition() Returns a tuple where the string is parted into three parts

replace() Returns a string where a specified value is replaced with a specified value

rfind() Searches the string for a specified value and returns the last position of where it was found

rindex() Searches the string for a specified value and returns the last position of where it was found

rjust() Returns a right justified version of the string

rpartition() Returns a tuple where the string is parted into three parts

rsplit() Splits the string at the specified separator, and returns a list

rstrip() Returns a right trim version of the string

split() Splits the string at the specified separator, and returns a list

splitlines() Splits the string at line breaks and returns a list

startswith() Returns true if the string starts with the specified value strip() Returns a trimmed version of the string

swapcase() Swaps cases, lower case becomes upper case and vice versa

title() Converts the first character of each word to upper case

translate() Returns a translated string

upper() Converts a string into upper case

zfill() Fills the string with a specified number of 0 values at the beginning

```
print(10 > 9)
print(10 == 9)
print(10 < 9)


print(bool("Hello"))
print(bool(15))

x = "Hello"
y = 15

print(bool(x))
print(bool(y))


bool("abc")
bool(123)
bool(["apple", "cherry", "banana"])


bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})


def myFunction() :
  return True

if myFunction():
  print("YES!")
else:
  print("NO!")


x = 200
print(isinstance(x, int))
```

```
True
False
False
True
True
True
True
YES!
True
```

Start coding or generate with AI.

Python Operators

Arithmetic operators

Assignment operators

Comparison operators

Logical operators

Identity operators

Membership operators

Bitwise operators

Start coding or generate with AI.

- Addition x + y
- Subtraction x - y
- Multiplication x * y

/ Division x / y

% Modulus x % y

** Exponentiation x ** y

// Floor division x // y

Start coding or generate with AI.

= x = 5 x = 5

+= x += 3 x = x + 3

-= x -= 3 x = x - 3

*= x *= 3 x = x * 3

/= x /= 3 x = x / 3

%= x %= 3 x = x % 3

//= x //= 3 x = x // 3

**= x **= 3 x = x ** 3

&= x &= 3 x = x & 3

|= x |= 3 x = x | 3

^= x ^= 3 x = x ^ 3

>>= x >>= 3 x = x >> 3

<<= x <<= 3 x = x << 3

:= print(x := 3) x = 3

print(x)

```
'''
==  Equal   x == y

!=  Not equal   x != y

>   Greater than    x > y
```

```
<   Less than   x < y

>=  Greater than or equal to    x >= y

<=  Less than or equal to   x <= y
'''
```

```
'\n==\tEqual\tx == y\t\n\n!=\tNot equal\tx != y\t\n\n>\tGreater than\tx > y\t\n\n<\tLess than\tx < y\t\n\n>=\tGreater than or equal to
```

Start coding or generate with AI.

```
"""
and     Returns True if both statements are true    x < 5 and  x < 10

or  Returns True if one of the statements is true   x < 5 or x < 4

not Reverse the result, returns False if the result is true not(x < 5 and x < 10)

is  Returns True if both variables are the same object  x is y

is not  Returns True if both variables are not the same object  x is not y

n   Returns True if a sequence with the specified value is present in the object    x in y

not in  Returns True if a sequence with the specified value is not present in the object    x

&   AND Sets each bit to 1 if both bits are 1   x & y

|   OR  Sets each bit to 1 if one of two bits is 1  x | y

^   XOR Sets each bit to 1 if only one of two bits is 1 x ^ y

~   NOT Inverts all the bits    ~x

<<  Zero fill left shift    Shift left by pushing zeros in from the right and let the leftmost bits fall off

>>  Signed right shift  Shift right by pushing copies of the leftmost bit in from the left, and let the right
"""
```

```
'\nand \tReturns True if both statements are true\tx < 5 and  x < 10\t\n\nor\tReturns True if one of the statements is true\tx < 5 or x
< 4\n\nnot\tReverse the result, returns False if the result is true\tnot(x < 5 and x < 10)\n\nis \tReturns True if both variables are t
he same object\tx is y\t\n\nis not\tReturns True if both variables are not the same object\tx is not y\n\nn \tReturns True if a sequenc
e with the specified value is present in the object\tx in y\t\n\nnot in\tReturns True if a sequence with the specified value is not pre
sent in the object\tx\n\n& \tAND\tSets each bit to 1 if both bits are 1\tx & y\t\n\n|\tOR\tSets each bit to 1 if one of two bits is 1\t
x | y\t\n\n^\tXOR\tSets each bit to 1 if only one of two bits is 1\tx ^ y\t\n\n~\tNOT\tInverts all the bits\t~x\t\n\n<<\tZero fill left
```

Start coding or generate with AI.

Operator Precedence

() Parentheses

** Exponentiation

+x -x ~x Unary plus, unary minus, and bitwise NOT

- / // % Multiplication, division, floor division, and modulus
- ◦ Addition and subtraction

<< >> Bitwise left and right shifts

& Bitwise AND

^ Bitwise XOR

| Bitwise OR

== != > >= < <= is is not in not in Comparisons, identity, and membership operators

not Logical NOT

and AND

or OR

```
print((6 + 3) - (6 + 3))

print(100 + 5 * 3)

print(5 + 4 - 7 + 3)
```

```
0
115
5
```

Start coding or generate with AI.

Python List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Allow Duplicates

Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)

thislist = ["apple", "banana", "cherry", "apple", "cherry"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
print(len(thislist))

list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]

list1 = ["abc", 34, True, 40, "male"]

mylist = ["apple", "banana", "cherry"]
print(type(mylist))

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets
print(thislist)
```

```
['apple', 'banana', 'cherry']
['apple', 'banana', 'cherry', 'apple', 'cherry']
3
<class 'list'>
```

```
['apple', 'banana', 'cherry']
```

Start coding or generate with AI.

Python Collections (Arrays)

There are four collection data types in the Python programming language:

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.

Dictionary is a collection which is ordered** and changeable. No duplicate members.

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])

thislist = ["apple", "banana", "cherry"]
print(thislist[-1])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])

thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")

thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
```

```
print(thislist)

thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)

thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

```
banana
cherry
['cherry', 'orange', 'kiwi']
['apple', 'banana', 'cherry', 'orange']
['cherry', 'orange', 'kiwi', 'melon', 'mango']
['orange', 'kiwi', 'melon']
Yes, 'apple' is in the fruits list
['apple', 'blackcurrant', 'cherry']
['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']
['apple', 'blackcurrant', 'watermelon', 'cherry']
['apple', 'watermelon']
['apple', 'banana', 'watermelon', 'cherry']
['apple', 'banana', 'cherry', 'orange']
['apple', 'orange', 'banana', 'cherry']
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
['apple', 'banana', 'cherry', 'kiwi', 'orange']
```

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)

thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)

thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)

thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)

thislist = ["apple", "banana", "cherry"]
del thislist

thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
['apple', 'cherry']
['apple', 'cherry', 'banana', 'kiwi']
['apple', 'cherry']
['apple', 'banana']
['banana', 'cherry']
[]
```

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)

thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
```

```
  print(thislist[i])

thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1

thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)


fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)

newlist = [x for x in fruits if x != "apple"]
print(newlist)
newlist = [x for x in fruits]
print(newlist)
newlist = [x for x in range(10)]
print(newlist)
newlist = [x for x in range(10) if x < 5]
print(newlist)
newlist = [x.upper() for x in fruits]
print(newlist)
newlist = ['hello' for x in fruits]
print(newlist)
newlist = [x if x != "banana" else "orange" for x in fruits]
print(newlist)
```

```
apple
banana
cherry
apple
banana
cherry
apple
banana
cherry
apple
banana
cherry
['apple', 'banana', 'mango']
['apple', 'banana', 'mango']
['banana', 'cherry', 'kiwi', 'mango']
['apple', 'banana', 'cherry', 'kiwi', 'mango']
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4]
['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']
['hello', 'hello', 'hello', 'hello', 'hello']
['apple', 'orange', 'cherry', 'kiwi', 'mango']
```

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)

thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)

def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)

thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)

thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)

thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
[23, 50, 65, 82, 100]
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
[100, 82, 65, 50, 23]
[50, 65, 23, 82, 100]
['Kiwi', 'Orange', 'banana', 'cherry']
['banana', 'cherry', 'Kiwi', 'Orange']
['cherry', 'Kiwi', 'Orange', 'banana']
```

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)

thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)

thislist = ["apple", "banana", "cherry"]
mylist = thislist[:]
print(mylist)

list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)

list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
['apple', 'banana', 'cherry']
['apple', 'banana', 'cherry']
['apple', 'banana', 'cherry']
['a', 'b', 'c', 1, 2, 3]
['a', 'b', 'c', 1, 2, 3]
['a', 'b', 'c', 1, 2, 3]
```

Start coding or generate with AI.

Python - List Methods

append() Adds an element at the end of the list

clear() Removes all the elements from the list

copy() Returns a copy of the list

count() Returns the number of elements with the specified value

extend() Add the elements of a list (or any iterable), to the end of the current list

index() Returns the index of the first element with the specified value

insert() Adds an element at the specified position

pop() Removes the element at the specified position

remove() Removes the item with the specified value

reverse() Reverses the order of the list

sort() Sorts the list

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)

thistuple = ("apple", "banana", "cherry", "apple", "cherry") #Allow duplicate
print(thistuple)

thistuple = ("apple", "banana", "cherry")
print(len(thistuple))

thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))

tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

tuple1 = ("abc", 34, True, 40, "male")

mytuple = ("apple", "banana", "cherry")
print(type(mytuple))

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])

thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])

thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")

x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)

thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)

thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y

print(thistuple)

thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)

thistuple = ("apple", "banana", "cherry")
#del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists

fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, *red) = fruits

print(green)
```

```
print(yellow)
print(red)


fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)
print(tropic)
print(red)

thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)


thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])


thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1


tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)


fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
('apple', 'banana', 'cherry')
('apple', 'banana', 'cherry', 'apple', 'cherry')
3
<class 'tuple'>
<class 'str'>
<class 'tuple'>
('apple', 'banana', 'cherry')
banana
cherry
('cherry', 'orange', 'kiwi')
('apple', 'banana', 'cherry', 'orange')
('cherry', 'orange', 'kiwi', 'melon', 'mango')
('orange', 'kiwi', 'melon')
Yes, 'apple' is in the fruits tuple
('apple', 'kiwi', 'cherry')
('apple', 'banana', 'cherry', 'orange')
('apple', 'banana', 'cherry')
apple
banana
cherry
apple
banana
['cherry', 'strawberry', 'raspberry']
apple
['mango', 'papaya', 'pineapple']
```

```
cherry
apple
banana
cherry
apple
banana
cherry
apple
banana
cherry
('a', 'b', 'c', 1, 2, 3)
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

Tuple Methods

Python has two built-in methods that you can use on tuples.

count() Returns the number of times a specified value occurs in a tuple

index() Searches the tuple for a specified value and returns the position of where it was found

Start coding or generate with AI.

Python Sets

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable*, and unindexed.

- Note: Set items are unchangeable, but you can remove items and add new items.

Sets are written with curly brackets.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)

thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)

thisset = {"apple", "banana", "cherry", True, 1, 2}

print(thisset)

thisset = {"apple", "banana", "cherry", False, True, 0}

print(thisset)

thisset = {"apple", "banana", "cherry"}

print(len(thisset))

set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}

set1 = {"abc", 34, True, 40, "male"}

myset = {"apple", "banana", "cherry"}
print(type(myset))

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)


thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)

thisset = {"apple", "banana", "cherry"}

print("banana" not in thisset)



thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)


thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)



thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)


thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)


thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)


thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

#del thisset

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

```
{'cherry', 'banana', 'apple'}
{'cherry', 'banana', 'apple'}
{True, 2, 'cherry', 'banana', 'apple'}
{False, True, 'cherry', 'banana', 'apple'}
3
<class 'set'>
{'cherry', 'banana', 'apple'}
cherry
banana
apple
True
False
{'cherry', 'banana', 'orange', 'apple'}
{'pineapple', 'mango', 'papaya', 'cherry', 'banana', 'apple'}
{'kiwi', 'apple', 'orange', 'cherry', 'banana'}
{'cherry', 'apple'}
{'cherry', 'apple'}
cherry
{'banana', 'apple'}
set()
{'cherry', 'banana', 'apple'}
cherry
banana
apple
```

Join Sets

There are several ways to join two or more sets in Python.

The union() and update() methods joins all items from both sets.

The intersection() method keeps ONLY the duplicates.

The difference() method keeps the items from the first set that are not in the other set(s).

The symmetric_difference() method keeps all items EXCEPT the duplicates.

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}

set3 = set1 | set2
print(set3)
```

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1.union(set2, set3, set4)
print(myset)
```

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}

myset = set1 | set2 | set3 |set4
print(myset)

x = {"a", "b", "c"}
y = (1, 2, 3)

z = x.union(y)
print(z)
```

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.intersection(set2)
print(set3)
```

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 & set2
print(set3)
```

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.intersection_update(set2)

print(set1)
```

```
set1 = {"apple", 1,  "banana", 0, "cherry"}
set2 = {False, "google", 1, "apple", 2, True}

set3 = set1.intersection(set2)

print(set3)
```

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
```

```
set3 = set1.difference(set2)

print(set3)


set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 - set2
print(set3)


set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.difference_update(set2)

print(set1)


set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1.symmetric_difference(set2)

print(set3)


set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set3 = set1 ^ set2
print(set3)



set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.symmetric_difference_update(set2)

print(set1)
```

```
{1, 'b', 2, 3, 'a', 'c'}
{1, 'b', 2, 3, 'a', 'c'}
{1, 2, 3, 'John', 'Elena', 'c', 'apple', 'bananas', 'b', 'a', 'cherry'}
{1, 2, 3, 'John', 'Elena', 'c', 'apple', 'bananas', 'b', 'a', 'cherry'}
{1, 2, 3, 'c', 'b', 'a'}
{1, 'b', 2, 3, 'a', 'c'}
{'apple'}
{'apple'}
{'apple'}
{False, 1, 'apple'}
{'cherry', 'banana'}
{'cherry', 'banana'}
{'cherry', 'banana'}
{'google', 'cherry', 'banana', 'microsoft'}
{'google', 'cherry', 'banana', 'microsoft'}
{'google', 'cherry', 'banana', 'microsoft'}
```

```
Start coding or generate with AI.
```

Python - Set Methods

add() Adds an element to the set

clear() Removes all the elements from the set

copy() Returns a copy of the set

difference() - Returns a set containing the difference between two or more sets

difference_update() -= Removes the items in this set that are also included in another, specified set

discard() Remove the specified item

intersection() & Returns a set, that is the intersection of two other sets

intersection_update() &= Removes the items in this set that are not present in other, specified set(s)

isdisjoint() Returns whether two sets have a intersection or not

issubset() <= Returns True if all items of this set is present in another set

```
<   Returns True if all items of this set is present in another, larger set
```

issuperset() >= Returns True if all items of another set is present in this set

```
>   Returns True if all items of another, smaller set is present in this set
```

pop() Removes an element from the set

remove() Removes the specified element

symmetric_difference() ^ Returns a set with the symmetric differences of two sets

symmetric_difference_update() ^= Inserts the symmetric differences from this set and another

union() | Return a set containing the union of sets

update() |= Update the set with the union of this set and others

```
Start coding or generate with AI.
```

Python Dictionaries

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)


print(len(thisdict))


thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))



thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]

x = thisdict.get("model")

x = thisdict.keys()

car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change

x = thisdict.values()

car = {
"brand": "Ford",
```

```
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change



car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change




x = thisdict.items()


car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change



car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
  thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
thisdict = {
```

```
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
#del thisdict["model"]
print(thisdict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
#del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)


for x in thisdict:
  print(x)

for x in thisdict:
  print(thisdict[x])


for x in thisdict.values():
  print(x)


for x in thisdict.keys():
  print(x)


for x, y in thisdict.items():
  print(x, y)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)


thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
Ford
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
3
<class 'dict'>
{'name': 'John', 'age': 36, 'country': 'Norway'}
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 2020])
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 1964, 'red'])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2020)])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])
Yes, 'model' is one of the keys in the thisdict dictionary
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
{'brand': 'Ford', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang'}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}


print(myfamily["child2"]["name"])

for x, obj in myfamily.items():
  print(x)
```

```
  for y in obj:
    print(y + ':', obj[y])
```

```
Tobias
child1
name: Emil
year: 2004
child2
name: Tobias
year: 2007
child3
name: Linus
year: 2011
```

Start coding or generate with AI.

Python Dictionary Methods

clear() Removes all the elements from the dictionary

copy() Returns a copy of the dictionary

fromkeys() Returns a dictionary with the specified keys and value

get() Returns the value of the specified key

items() Returns a list containing a tuple for each key value pair

keys() Returns a list containing the dictionary's keys

pop() Removes the element with the specified key

popitem() Removes the last inserted key-value pair

setdefault() Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

update() Updates the dictionary with the specified key-value pairs

values() Returns a list of all the values in the dictionary

Start coding or generate with AI.

Python Conditions

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
a = 33
b = 200
if b > a:
```

```
  print("b is greater than a") # you will get an error id indent move
```

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
if a > b: print("a is greater than b")
```

```
a = 2
b = 330
print("A") if a > b else print("B")
```

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
a = 33
b = 200
if not a > b:
```

```
  print("a is NOT greater than b")
```

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
a = 33
b = 200

if b > a:
  pass
```

```
day = 4
match day:
  case 1:
    print("Monday")
  case 2:
    print("Tuesday")
  case 3:
    print("Wednesday")
  case 4:
    print("Thursday")
  case 5:
    print("Friday")
  case 6:
    print("Saturday")
  case 7:
    print("Sunday")
```

```
day = 4
match day:
  case 6:
    print("Today is Saturday")
  case 7:
    print("Today is Sunday")
  case _:
    print("Looking forward to the Weekend")
```

```
day = 4
match day:
  case 1 | 2 | 3 | 4 | 5:
    print("Today is a weekday")
  case 6 | 7:
    print("I love weekends!")
```

```
month = 5
day = 4
match day:
  case 1 | 2 | 3 | 4 | 5 if month == 4:
```

```
      print("A weekday in April")
    case 1 | 2 | 3 | 4 | 5 if month == 5:
      print("A weekday in May")
    case _:
      print("No match")


i = 1
while i < 6:
  print(i)
  i += 1


i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1


i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)


i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")


fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)


for x in "banana":
  print(x)


fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break


fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)


fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
```

```
    continue
  print(x)


for x in range(6):
  print(x)



for x in range(2, 6):
  print(x)


for x in range(2, 30, 3):
  print(x)


for x in range(6):
  print(x)
else:
  print("Finally finished!")


for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")


adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)


for x in [0, 1, 2]:
  pass
```

```
2
5
8
11
14
17
20
23
26
29
0
1
2
3
4
5
Finally finished!
0
1
2
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

```
def my_function():
  print("Hello from a function")
```

```
def my_function():
  print("Hello from a function")

my_function()
```

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Test")
```

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")


def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")



def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)


def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))


def myfunction():
  pass


def my_function(x, /):
  print(x)

my_function(3)


def my_function(x):
  print(x)

my_function(x = 3)



def my_function(*, x):
  print(x)

my_function(x = 3)


def my_function(x):
  print(x)

my_function(3)

def my_function(*, x):
```

```
  print(x)

my_function( x = 3)

def my_function(a, b, /, *, c, d):
  print(a + b + c + d)

my_function(5, 6, c = 7, d = 8)



def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("Recursion Example Results:")
tri_recursion(6)
```

```
Hello from a function
Emil Refsnes
Tobias Refsnes
Linus Refsnes
Emil Refsnes
Emil Test
The youngest child is Linus
The youngest child is Linus
His last name is Refsnes
I am from Sweden
I am from India
I am from Norway
I am from Brazil
apple
banana
cherry
15
25
45
3
3
3
3
3
26
Recursion Example Results:
1
3
6
10
15
21
21
```

```
x = lambda a : a + 10
print(x(5))

x = lambda a, b : a * b
print(x(5, 6))

x = lambda a, b, c : a + b + c
print(x(5, 6, 2))


def myfunc(n):
  return lambda a : a * n
```

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))



def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))




def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))



car1 = "Ford"
car2 = "Volvo"
car3 = "BMW"



cars = "Toyota"

x = cars[0]

x = len(cars)

for x in cars:
  print(x)


# cars.append("Honda")

# cars.pop(1)

# cars.remove("Volvo")
```

```
15
30
13
22
33
22
33
T
o
y
o
t
a
```

```
print("Love you " * 10, "Python " * 10)
```

```
Love you Love you Love you Love you Love you Love you Love you Love you Love you Love you  Python Python Python Python Python Python Pyt
```

Start coding or generate with AI.