All about Python

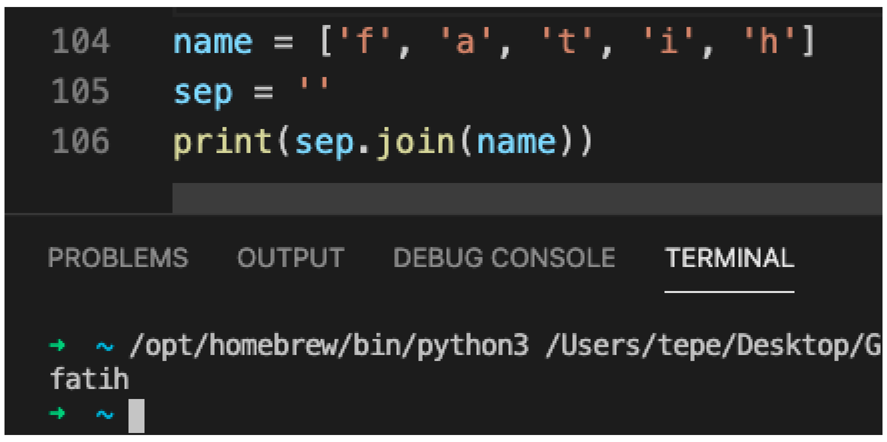
prepared by fatih tepe (click to go GitHub account)

join() method:

The `join()` method takes all items in an iterable and joins them into one string.

A string must be specified as the separator.

```
numList = ['1', '2', '3', '4']
separator = '$ '
print(separator.join(numList))

output
1$, 2$, 3$, 4
```



```
name = ['f', 'a', 't', 'i', 'h']
sep = ''
print(sep.join(name))
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/G
fatih
➜  ~ 
```

The input() function always returns a string.

If we want to ask the user to input an integer number, then we will need to convert the string returned from the input() function into an int.

Divide the integer

For example, if we divide the integer 100 by 20 then the result you might reasonably expect to produce might be 5; but it is not, it is actually 5.0:

print(100 / 20)
print(type(100 / 20))

The output is

5.0
<class 'float'>

x+=1

has the same behaviour as x = x + 1

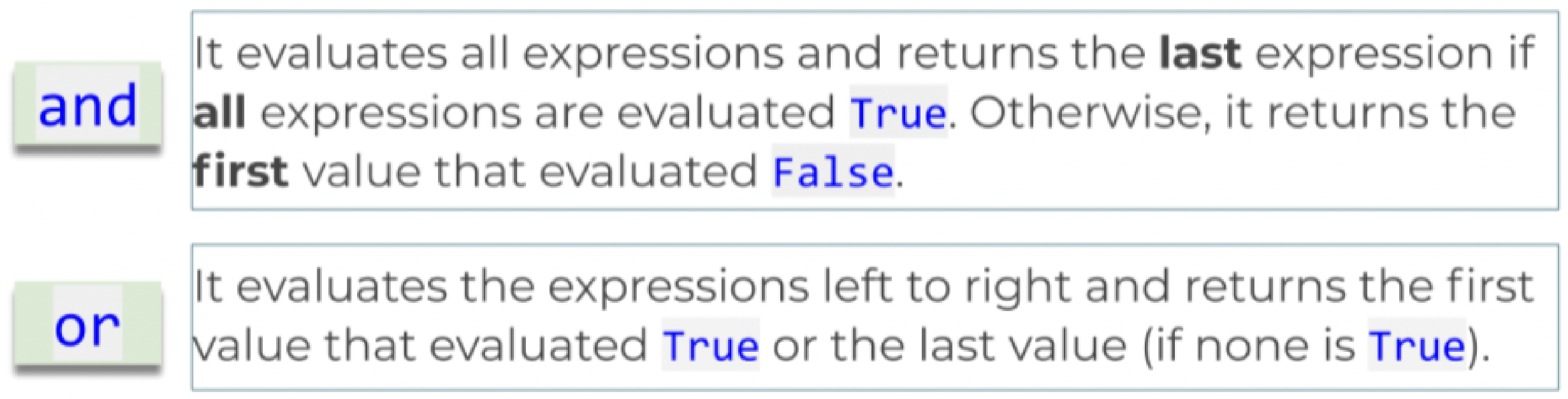
An alternative is an if expression.

The format of an if expression is

<result1> if <condition-is-met> else <result2>

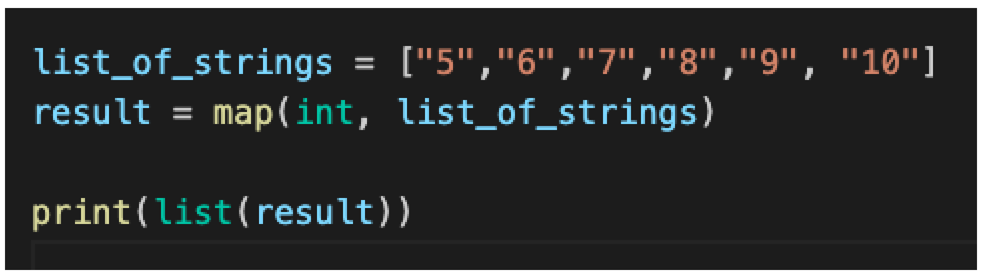
status = ('teenager' **if** age > 12 **and** age < 20 **else** 'not teenager')
print(status)

Boolean Logic Expressions



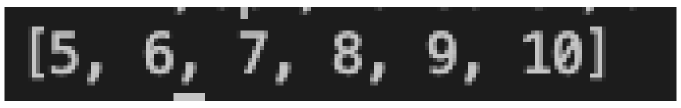
and	It evaluates all expressions and returns the **last** expression if **all** expressions are evaluated `True`. Otherwise, it returns the **first** value that evaluated `False`.
or	It evaluates the expressions left to right and returns the first value that evaluated `True` or the last value (if none is `True`).

MAP



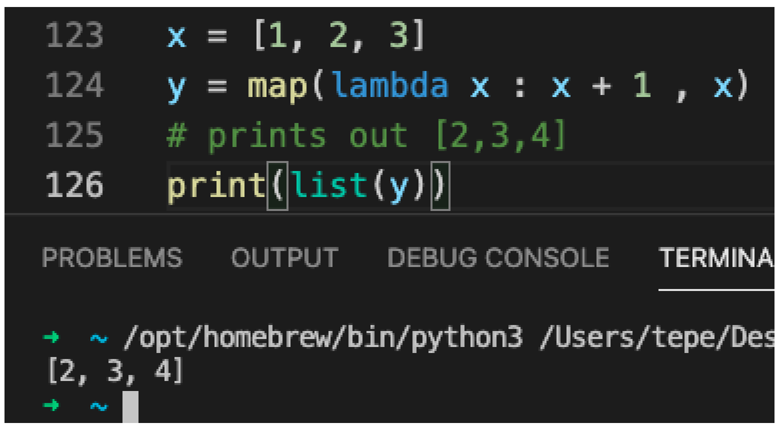
```python
list_of_strings = ["5","6","7","8","9", "10"]
result = map(int, list_of_strings)

print(list(result))
```

output:



```
[5, 6, 7, 8, 9, 10]
```

Python supports functional programming through a number of inbuilt features. One of the most useful is the map() function — especially in combination with lambda functions.



```python
x = [1, 2, 3]
y = map(lambda x : x + 1 , x)
# prints out [2,3,4]
print(list(y))
```

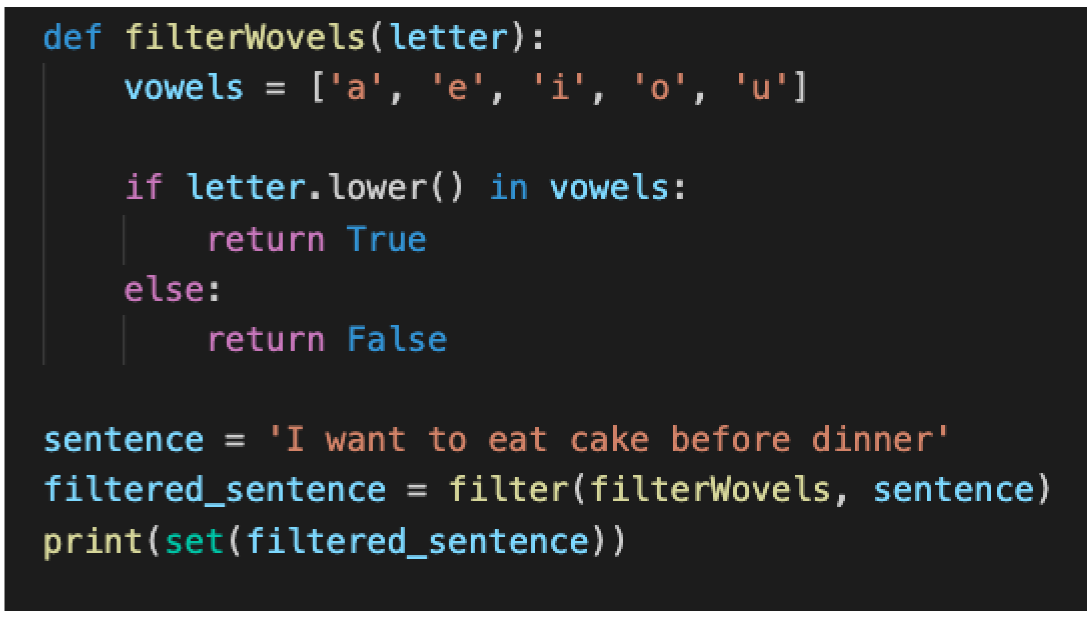
```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Des
[2, 3, 4]
➜  ~ 
```

In the example above, map() applies a simple lambda function to each element in x. It returns a map object, which can be converted to some iterable object such as a list or tuple.

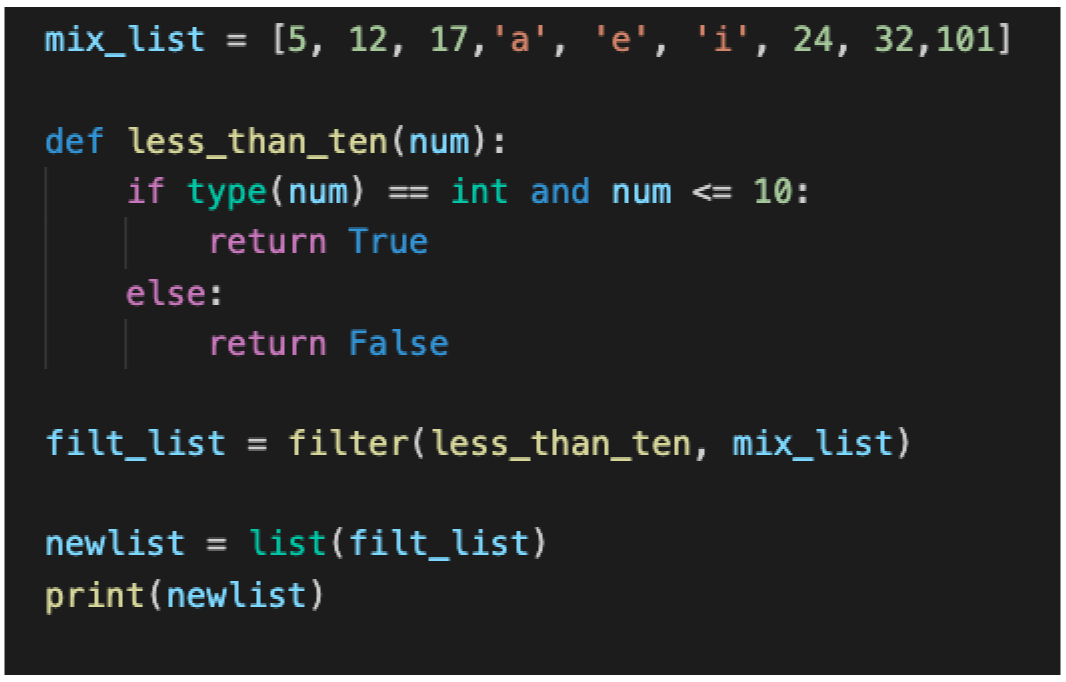
FILTER & FUNCTIONS

There are two type of functions

1. Perform a task (print)
2. Return a value(return)

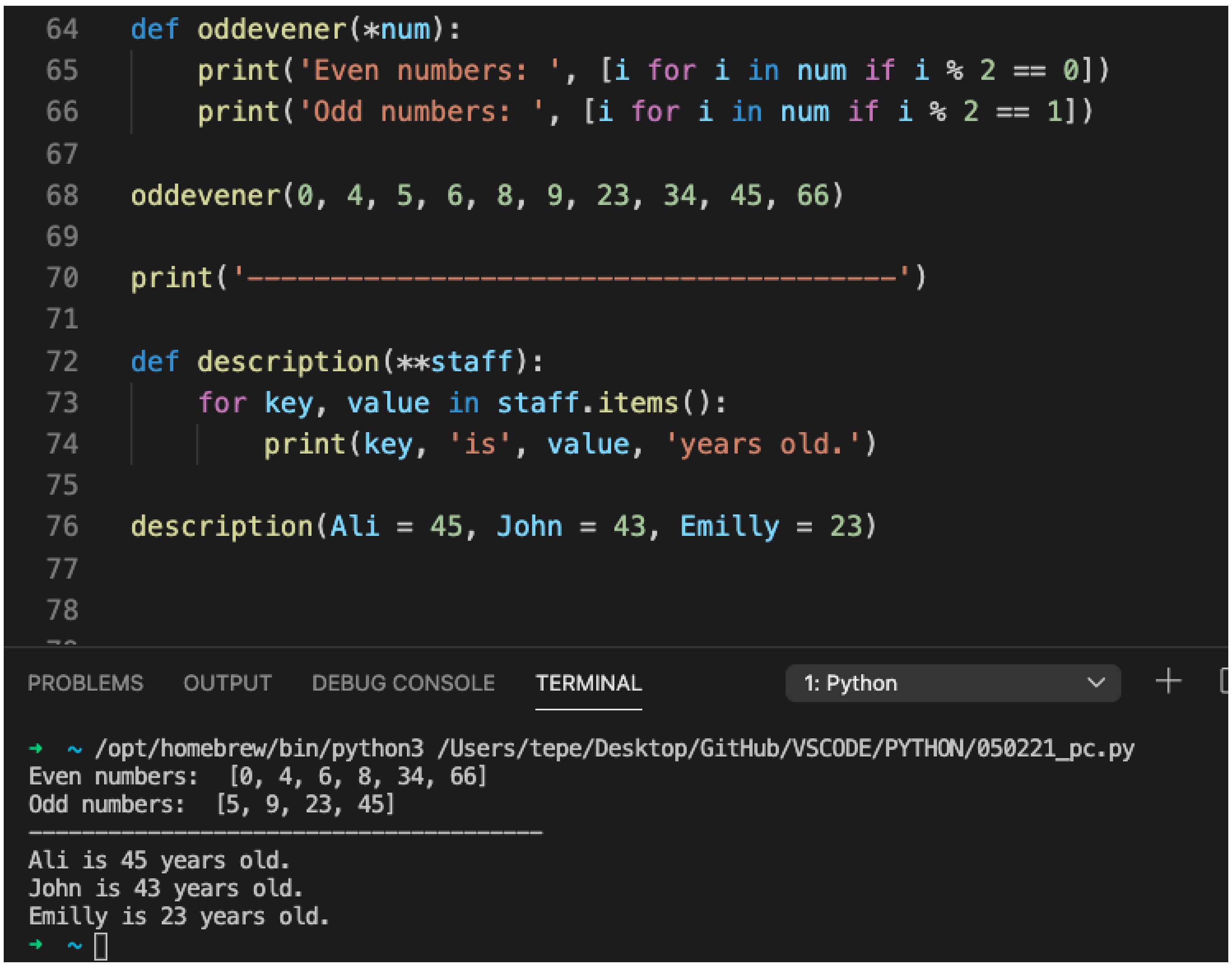


```python
def filterWovels(letter):
    vowels = ['a', 'e', 'i', 'o', 'u']

    if letter.lower() in vowels:
        return True
    else:
        return False

sentence = 'I want to eat cake before dinner'
filtered_sentence = filter(filterWovels, sentence)
print(set(filtered_sentence))
```



```python
mix_list = [5, 12, 17,'a', 'e', 'i', 24, 32,101]

def less_than_ten(num):
    if type(num) == int and num <= 10:
        return True
    else:
        return False

filt_list = filter(less_than_ten, mix_list)

newlist = list(filt_list)
print(newlist)
```

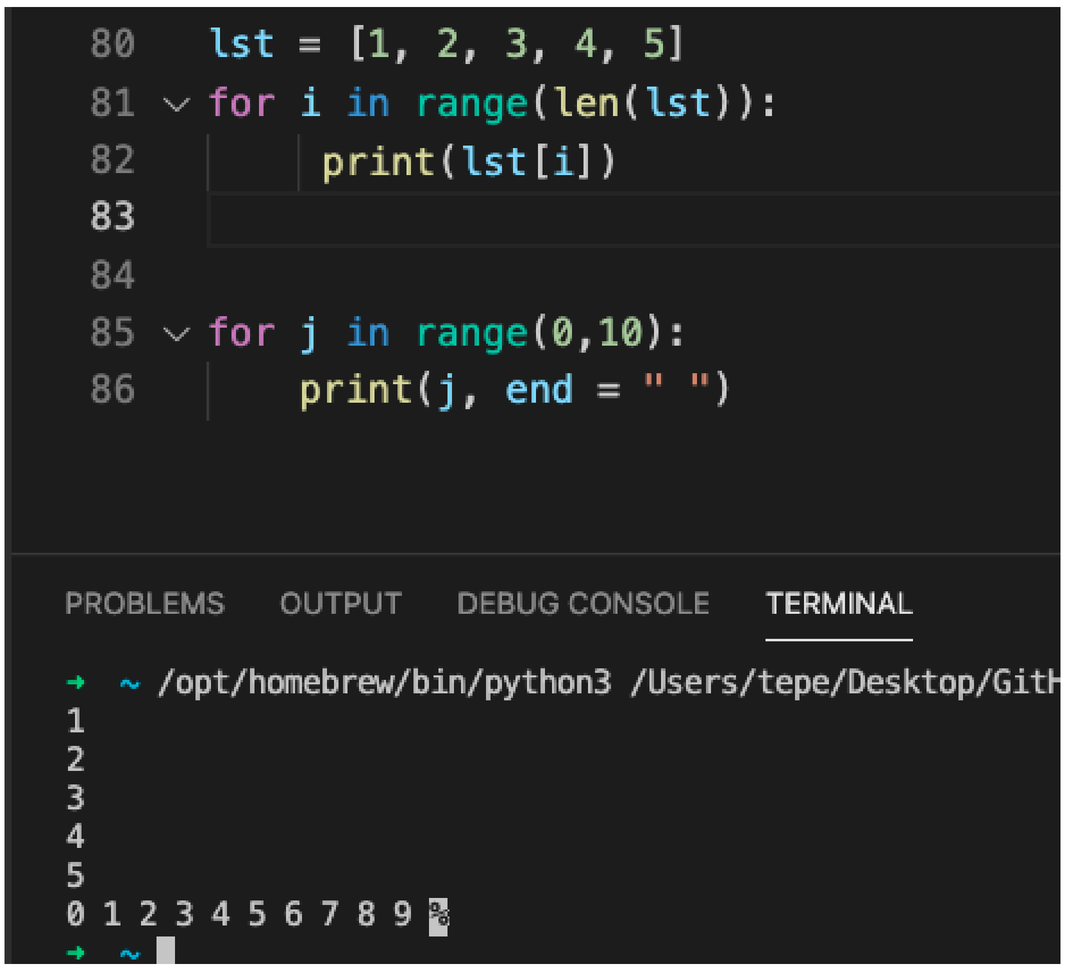
examples:



```python
def oddevener(*num):
    print('Even numbers: ', [i for i in num if i % 2 == 0])
    print('Odd numbers: ', [i for i in num if i % 2 == 1])

oddevener(0, 4, 5, 6, 8, 9, 23, 34, 45, 66)

print('---------------------------------------')

def description(**staff):
    for key, value in staff.items():
        print(key, 'is', value, 'years old.')

description(Ali = 45, John = 43, Emilly = 23)
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTHON/050221_pc.py
Even numbers:  [0, 4, 6, 8, 34, 66]
Odd numbers:  [5, 9, 23, 45]
---------------------------------------
Ali is 45 years old.
John is 43 years old.
Emilly is 23 years old.
➜  ~ 
```

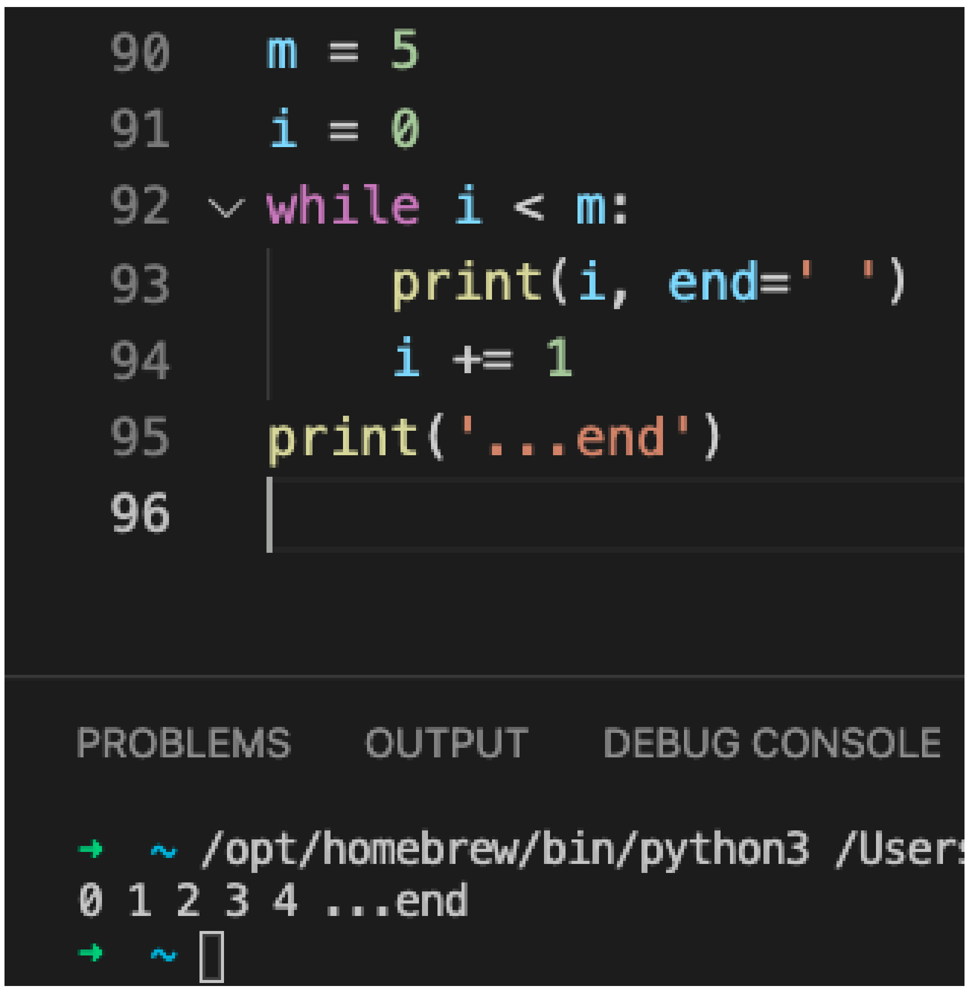
for loop: A for loop is used to iterate over a sequence that is either a list, tuple, dictionary, or a set. We can execute a set of statements once for each item in a list, tuple, or dictionary.



```python
lst = [1, 2, 3, 4, 5]
for i in range(len(lst)):
    print(lst[i])


for j in range(0,10):
    print(j, end = " ")
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitH
1
2
3
4
5
0 1 2 3 4 5 6 7 8 9 %
➜  ~
```

while loop: In Python, while loops are used to execute a block of statements repeatedly until a given condition is satisfied. Then, the expression is checked again and, if it is still true, the body is executed again. This continues until the expression becomes false.



```python
m = 5
i = 0
while i < m:
    print(i, end=' ')
    i += 1
print('...end')
```

```
➜  ~ /opt/homebrew/bin/python3 /User
0 1 2 3 4 ...end
➜  ~
```

The while loop needs a “loop condition.” If it stays True, it continues iterating.

DIFFERENCE BETWEEN PARAMATER AND ARGUMENT

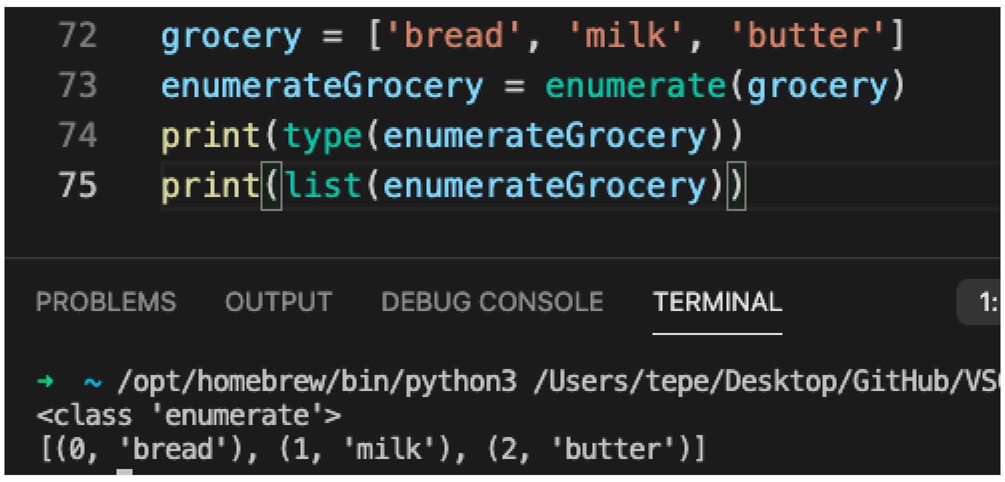
parameter = 'argument'

parameter is the input that define your funciton.

argument is the actual value for given parameter.

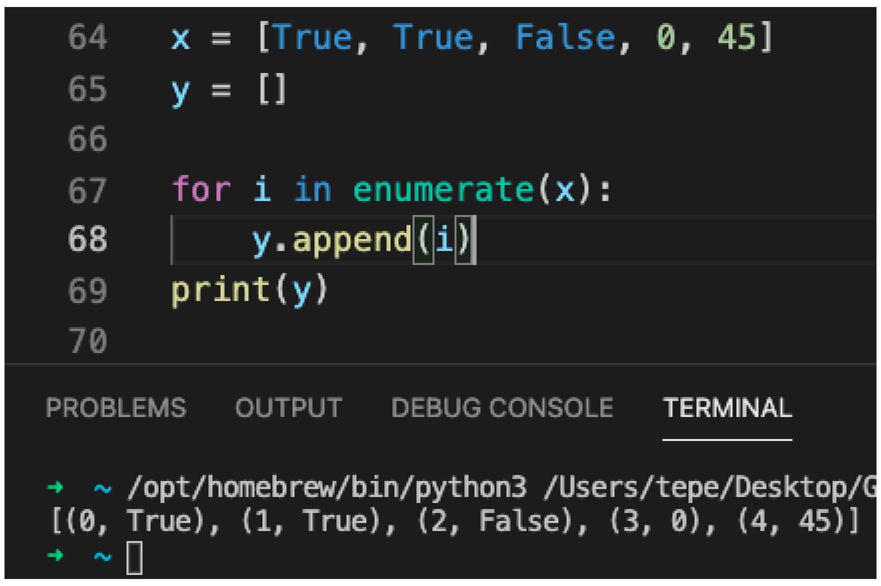
enumerate()

The enumerate() method adds counter to an iterable and returns it (the enumerate object).



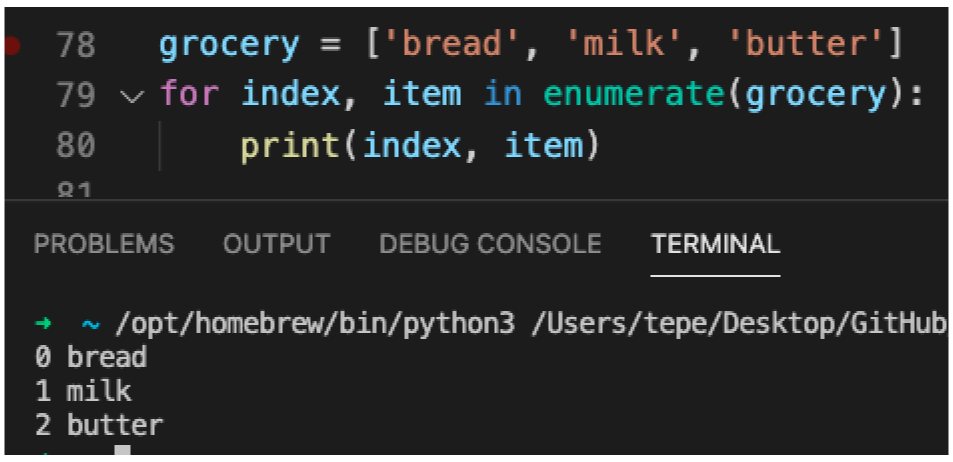
```python
grocery = ['bread', 'milk', 'butter']
enumerateGrocery = enumerate(grocery)
print(type(enumerateGrocery))
print(list(enumerateGrocery))
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VS
<class 'enumerate'>
[(0, 'bread'), (1, 'milk'), (2, 'butter')]
```



```python
x = [True, True, False, 0, 45]
y = []

for i in enumerate(x):
    y.append(i)
print(y)

```

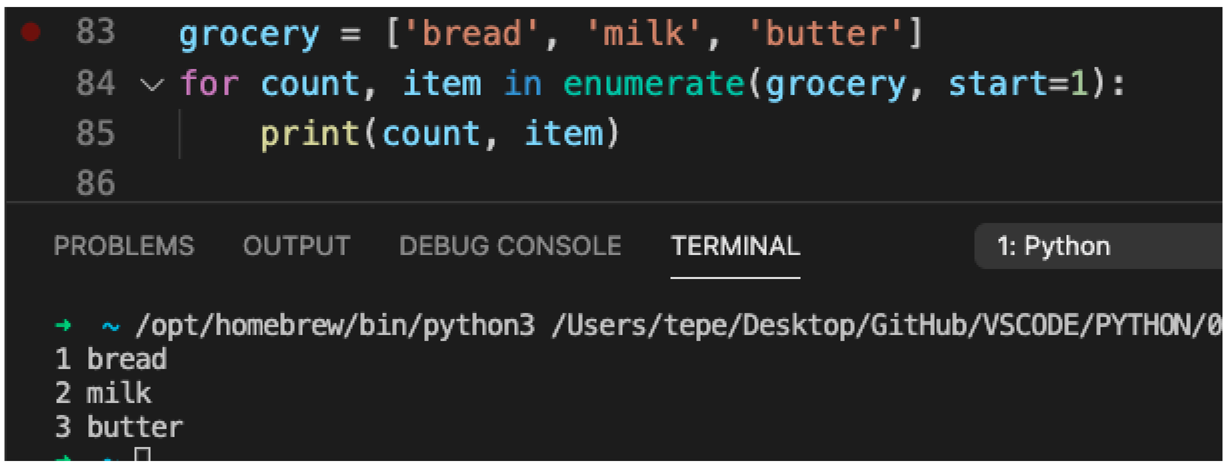
```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/G
[(0, True), (1, True), (2, False), (3, 0), (4, 45)]
➜  ~ 
```



```python
grocery = ['bread', 'milk', 'butter']
for index, item in enumerate(grocery):
    print(index, item)
```

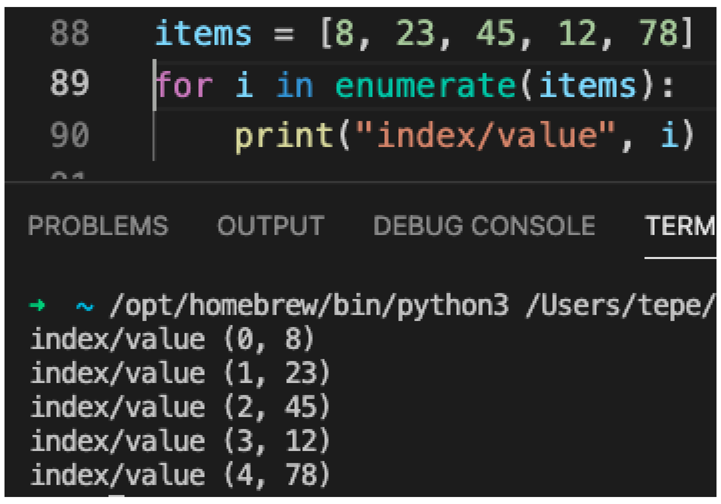
```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub
0 bread
1 milk
2 butter
```

And note that Python's indexes start at zero, so you would get 0 to 3 with the above. If you want the count, 1 to 4, do this:



```python
grocery = ['bread', 'milk', 'butter']
for count, item in enumerate(grocery, start=1):
    print(count, item)
```

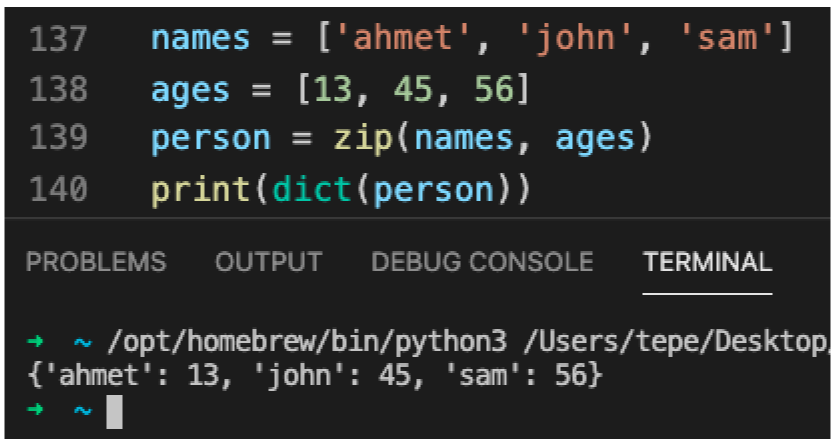
```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTHON/0
1 bread
2 milk
3 butter
```



```python
items = [8, 23, 45, 12, 78]
for i in enumerate(items):
    print("index/value", i)
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/
index/value (0, 8)
index/value (1, 23)
index/value (2, 45)
index/value (3, 12)
index/value (4, 78)
```

zip



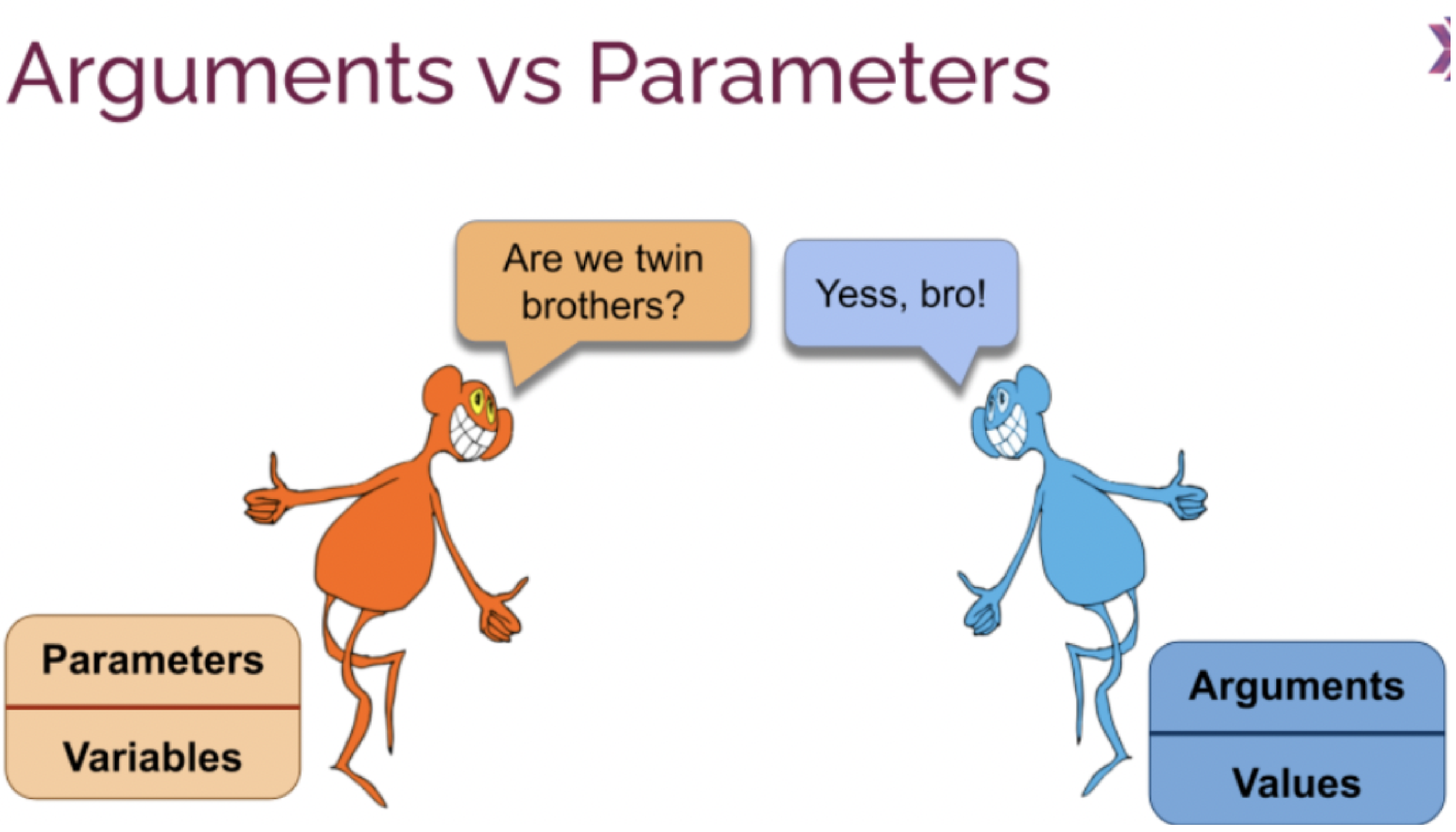
```python
names = ['ahmet', 'john', 'sam']
ages = [13, 45, 56]
person = zip(names, ages)
print(dict(person))
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop
{'ahmet': 13, 'john': 45, 'sam': 56}
➜  ~ 
```

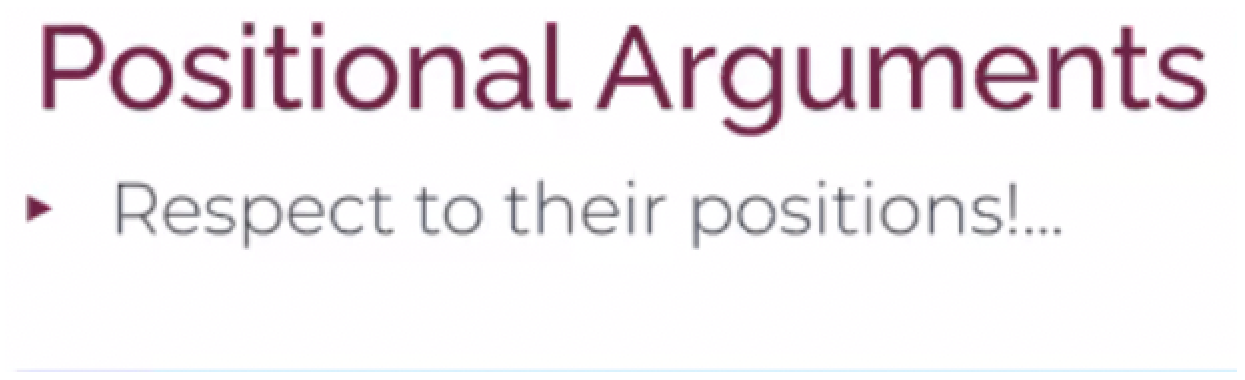
“An iterator:

is an object that can be iterated (looped) upon. It is used to abstract a container of data to make it behave like an iterable object. You probably already use a few iterable objects every day: strings, lists, and dictionaries to name a few.”

The Matter of Arguments



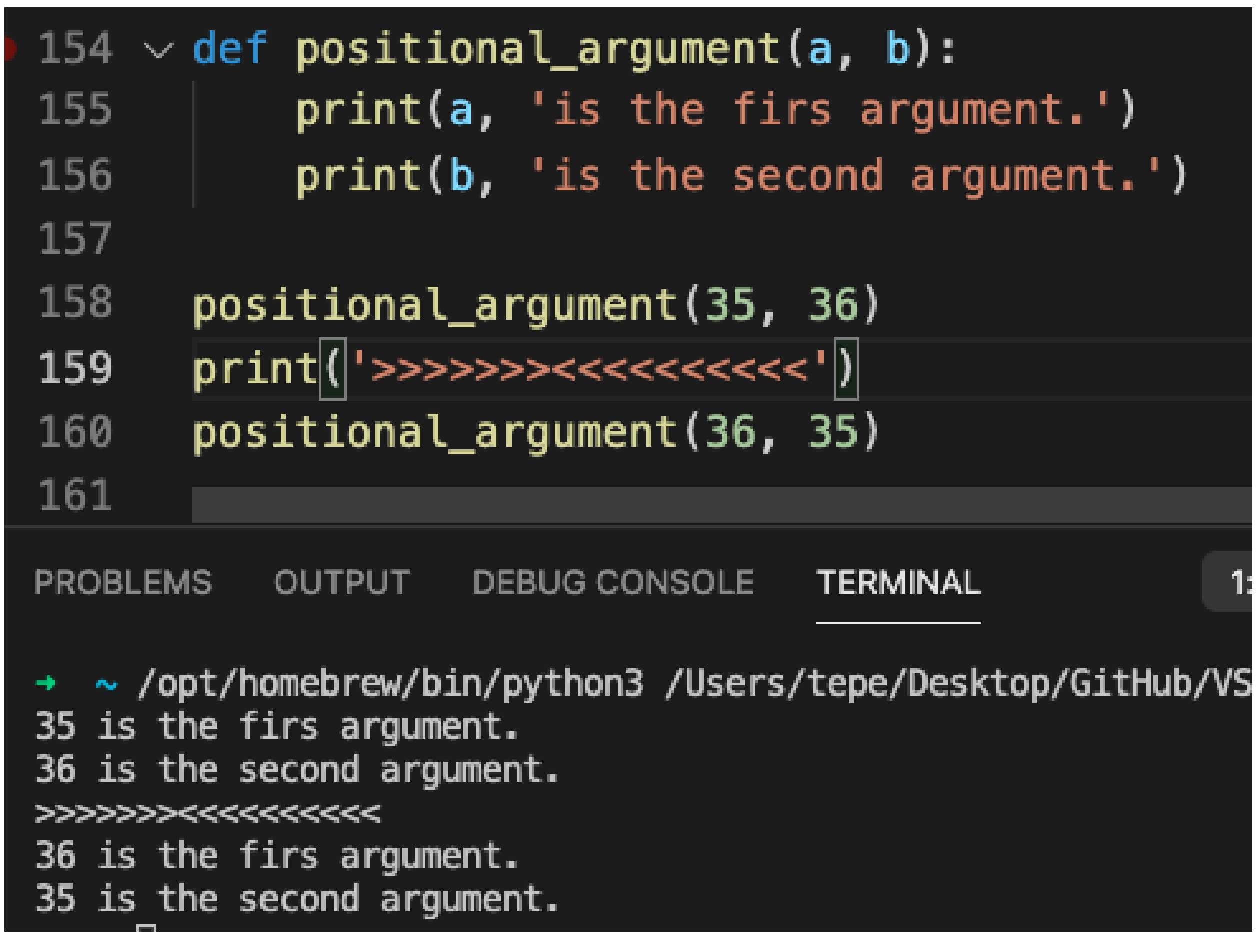
Arguments vs Parameters



Positional Arguments

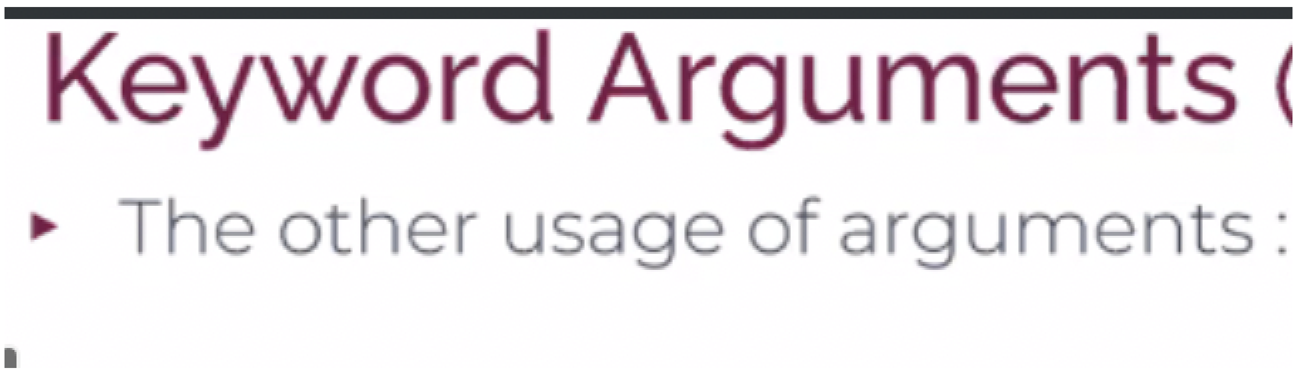
- Respect to their positions!...

When calling a function with **positional arguments**, they must be passed in ***order from left to right***



```python
def positional_argument(a, b):
    print(a, 'is the firs argument.')
    print(b, 'is the second argument.')

positional_argument(35, 36)
print('>>>>>>><<<<<<<<<<')
positional_argument(36, 35)
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VS
35 is the firs argument.
36 is the second argument.
>>>>>>><<<<<<<<<<
36 is the firs argument.
35 is the second argument.
```

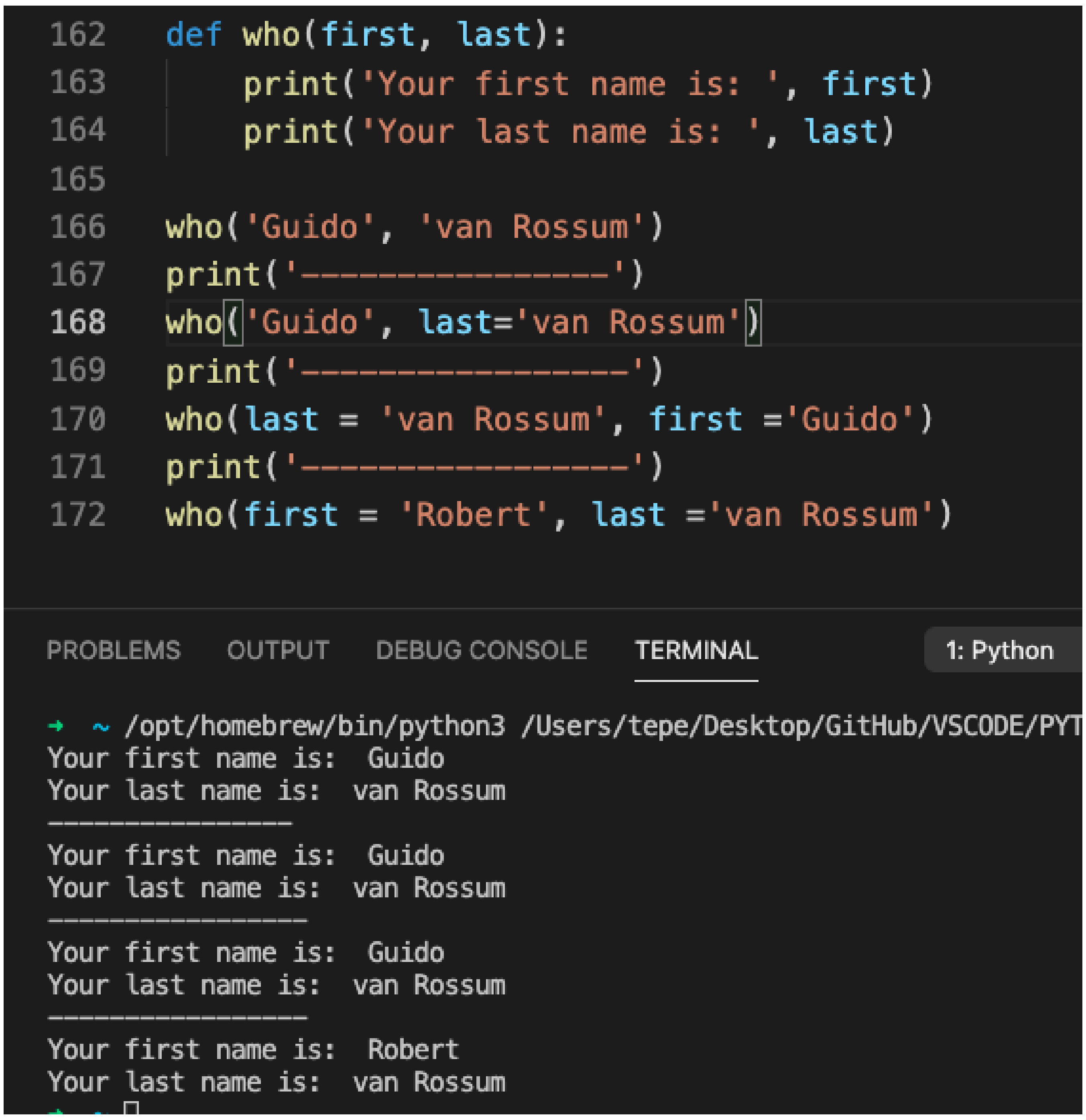


Keyword Arguments (

- The other usage of arguments :

Commonly and traditionally, ***kwargs*** is used as an abbreviation of ***keyword arguments***

> The formula syntax is : `kwargs=values`.



```python
def who(first, last):
    print('Your first name is: ', first)
    print('Your last name is: ', last)

who('Guido', 'van Rossum')
print('----------------')
who('Guido', last='van Rossum')
print('-----------------')
who(last = 'van Rossum', first ='Guido')
print('-----------------')
who(first = 'Robert', last ='van Rossum')
```

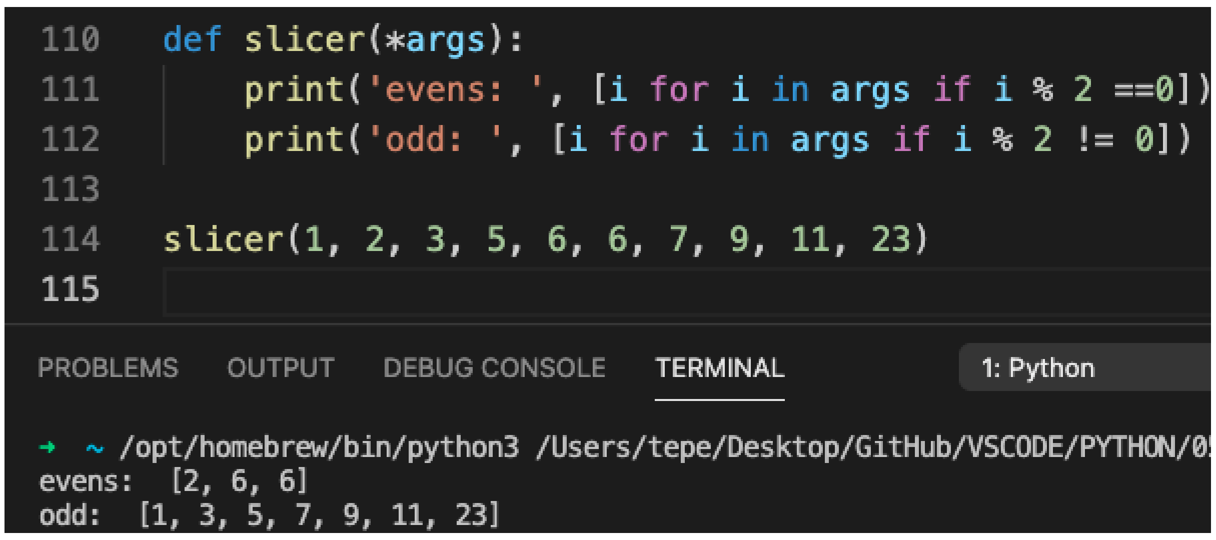
```
→ ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYT
Your first name is:  Guido
Your last name is:  van Rossum
----------------
Your first name is:  Guido
Your last name is:  van Rossum
-----------------
Your first name is:  Guido
Your last name is:  van Rossum
-----------------
Your first name is:  Robert
Your last name is:  van Rossum
```

Arbitrary Number of Arguments

> **Q**: What does this mean: `*args`, `**kwargs`? And why would we use it?
> **A**: We use `*args` when we aren't sure how many arguments are going to be passed to a function, or if we want to pass a stored list or tuple of arguments to a function. `**kwargs` is used when we don't know how many keyword arguments will be passed to a function, or it can be used to pass the values of a dictionary as keyword arguments. The identifiers args and kwargs are a convention, you could also use *bob and **billy but that would not be wise.
>
> - **Interview Q&A**

*args

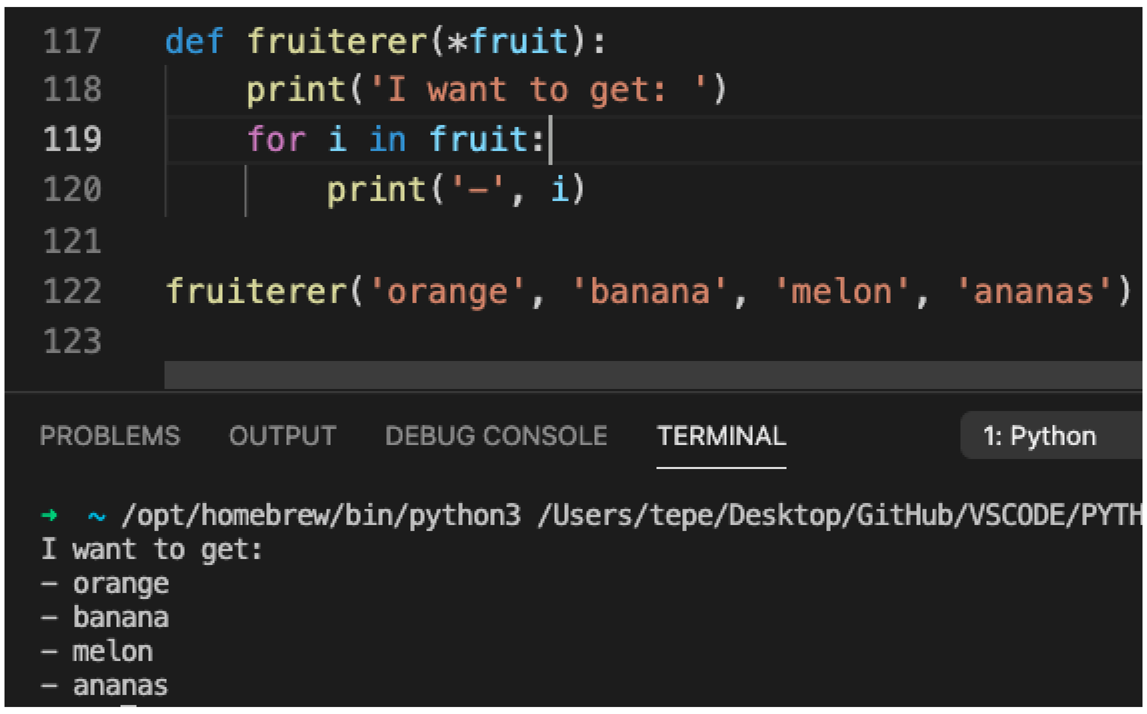
```
def name(*parameter)

    name(multiple args)
```



```python
def slicer(*args):
    print('evens: ', [i for i in args if i % 2 ==0])
    print('odd: ', [i for i in args if i % 2 != 0])

slicer(1, 2, 3, 5, 6, 6, 7, 9, 11, 23)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Python

```
➜ ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTHON/0
evens:  [2, 6, 6]
odd:  [1, 3, 5, 7, 9, 11, 23]
```

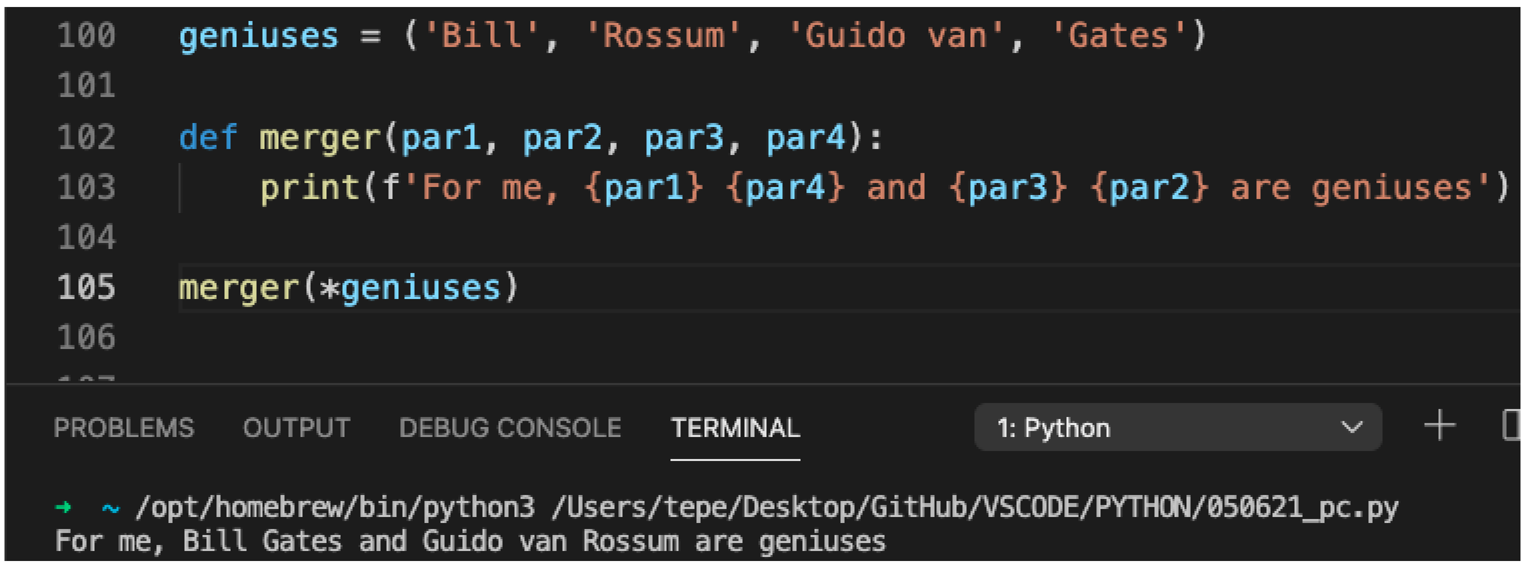


```python
def fruiterer(*fruit):
    print('I want to get: ')
    for i in fruit:
        print('-', i)

fruiterer('orange', 'banana', 'melon', 'ananas')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Python

```
➜ ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTH
I want to get:
- orange
- banana
- melon
- ananas
```

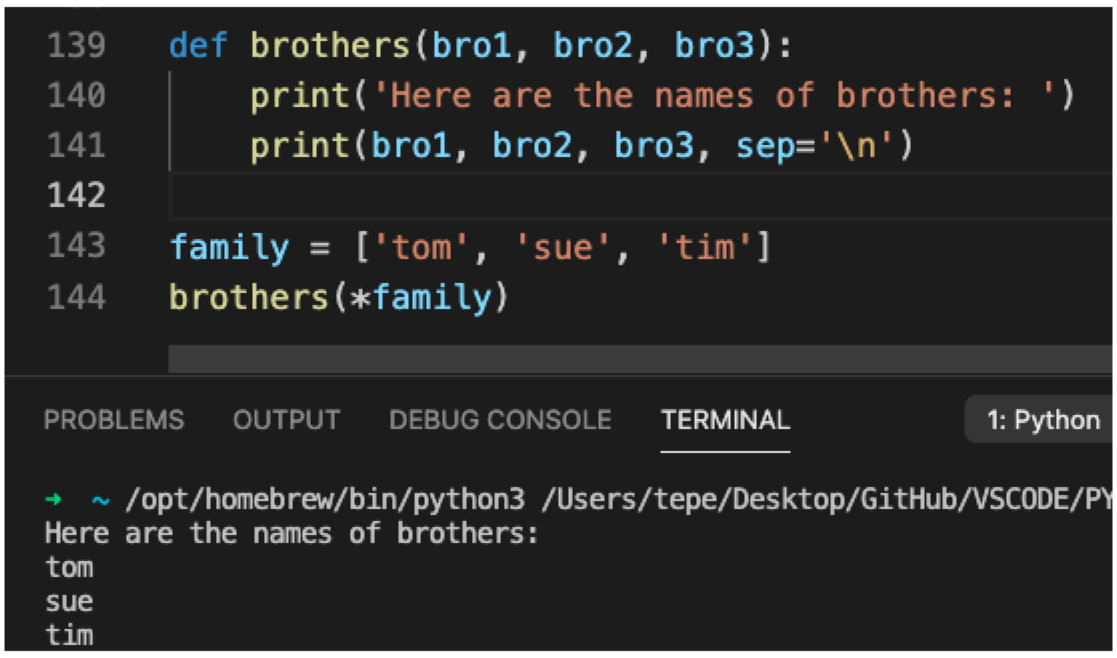
```
def name(multiple parameters)

    name(*variable)
```



```python
geniuses = ('Bill', 'Rossum', 'Guido van', 'Gates')

def merger(par1, par2, par3, par4):
    print(f'For me, {par1} {par4} and {par3} {par2} are geniuses')

merger(*geniuses)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Python

```
➜ ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTHON/050621_pc.py
For me, Bill Gates and Guido van Rossum are geniuses
```



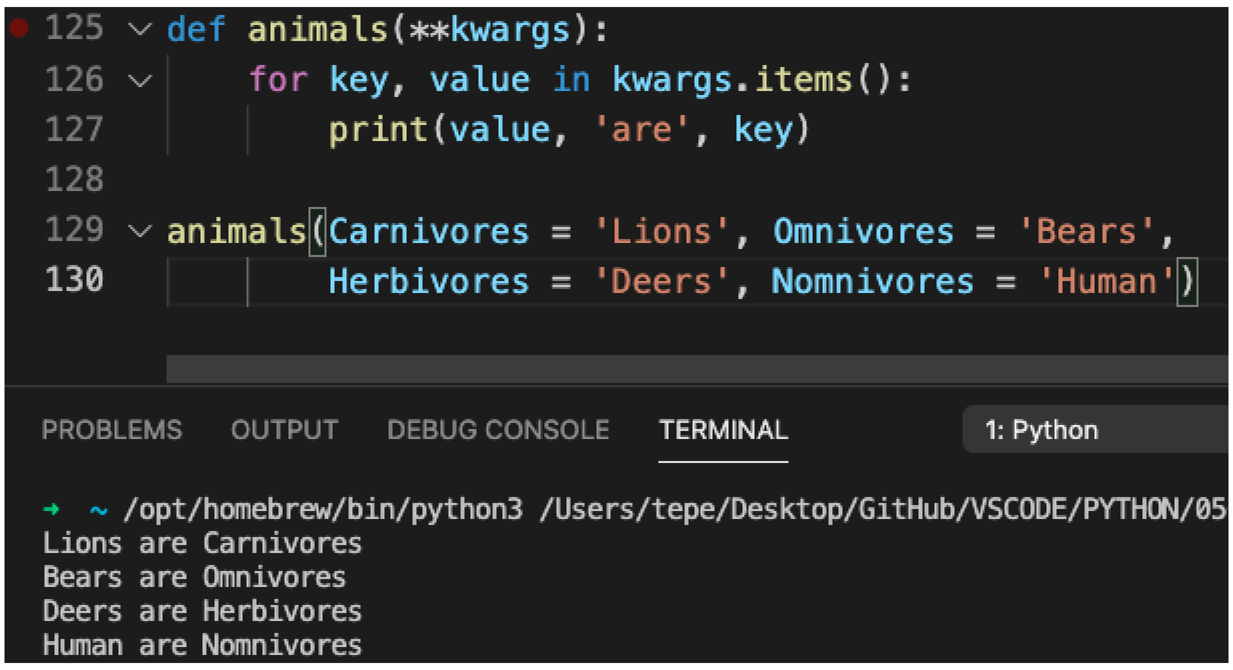
```python
def brothers(bro1, bro2, bro3):
    print('Here are the names of brothers: ')
    print(bro1, bro2, bro3, sep='\n')

family = ['tom', 'sue', 'tim']
brothers(*family)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Python

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PY
Here are the names of brothers:
tom
sue
tim
```

**kwargs

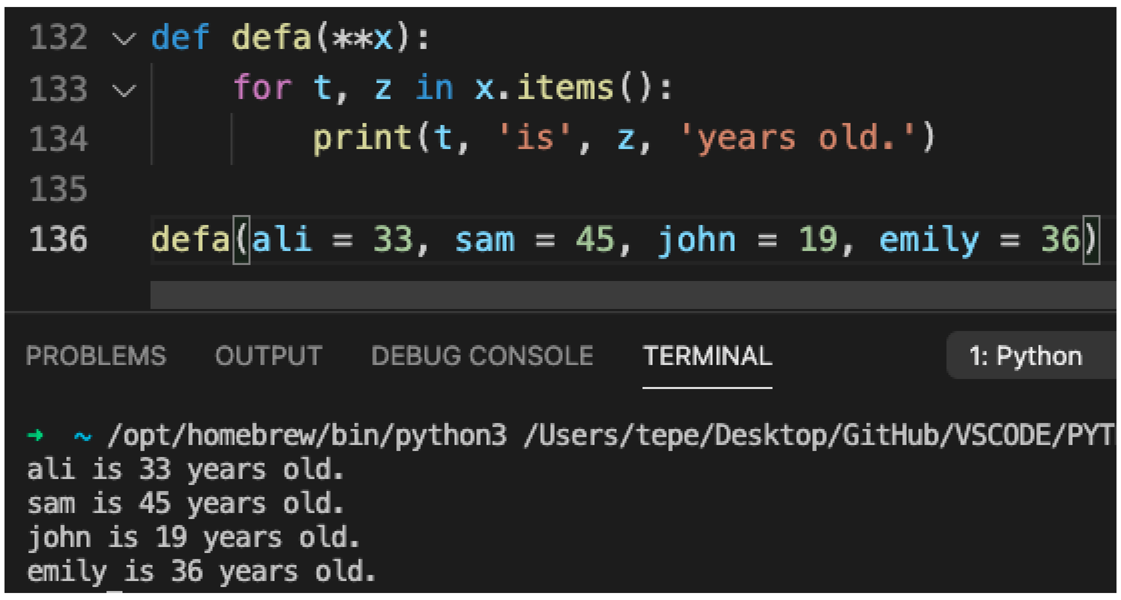
```
def name(**parameter)

    name(multiple kwargs)
```



```python
def animals(**kwargs):
    for key, value in kwargs.items():
        print(value, 'are', key)

animals(Carnivores = 'Lions', Omnivores = 'Bears',
        Herbivores = 'Deers', Nomnivores = 'Human')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Python

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTHON/05
Lions are Carnivores
Bears are Omnivores
Deers are Herbivores
Human are Nomnivores
```

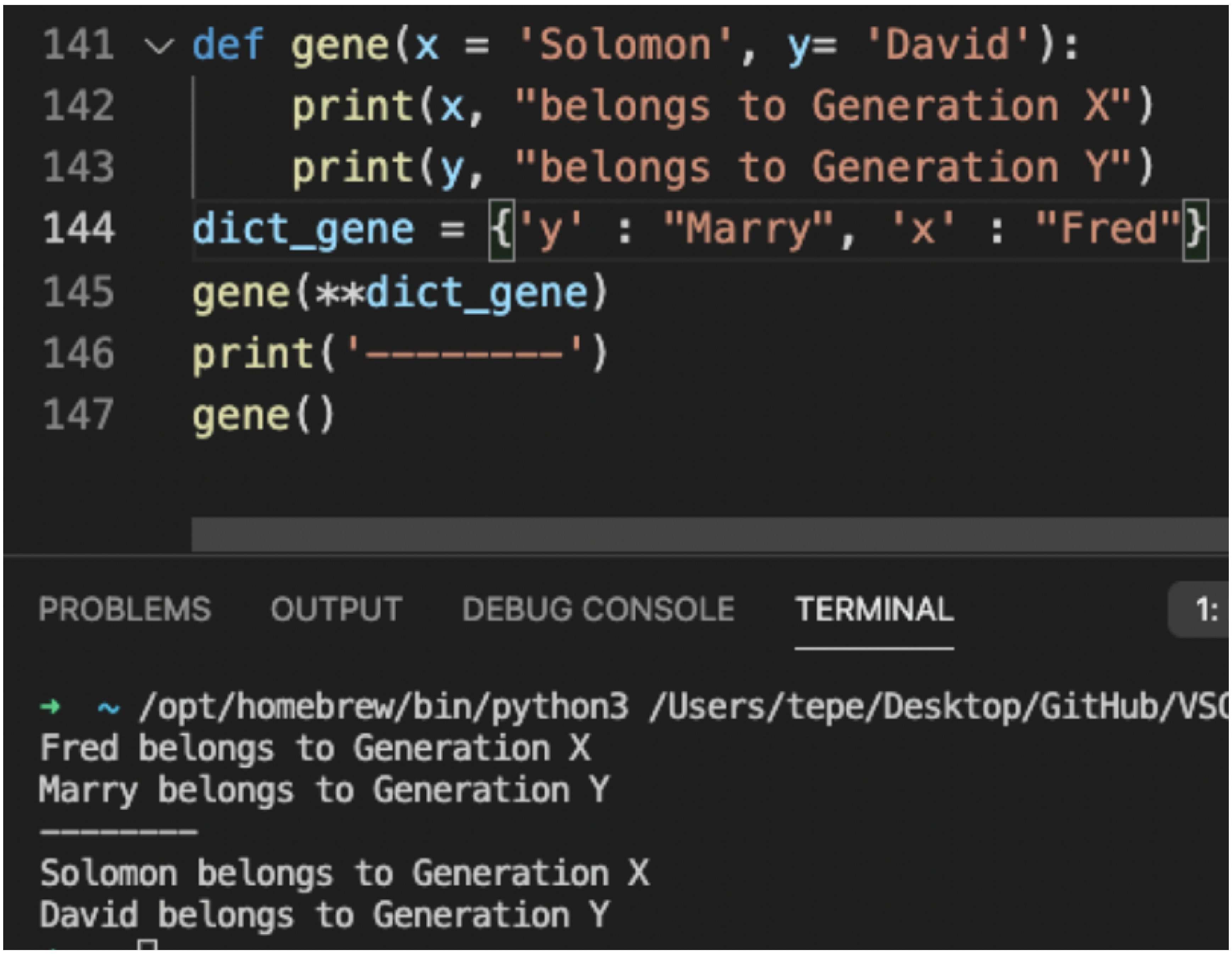


```python
def defa(**x):
    for t, z in x.items():
        print(t, 'is', z, 'years old.')

defa(ali = 33, sam = 45, john = 19, emily = 36)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Python

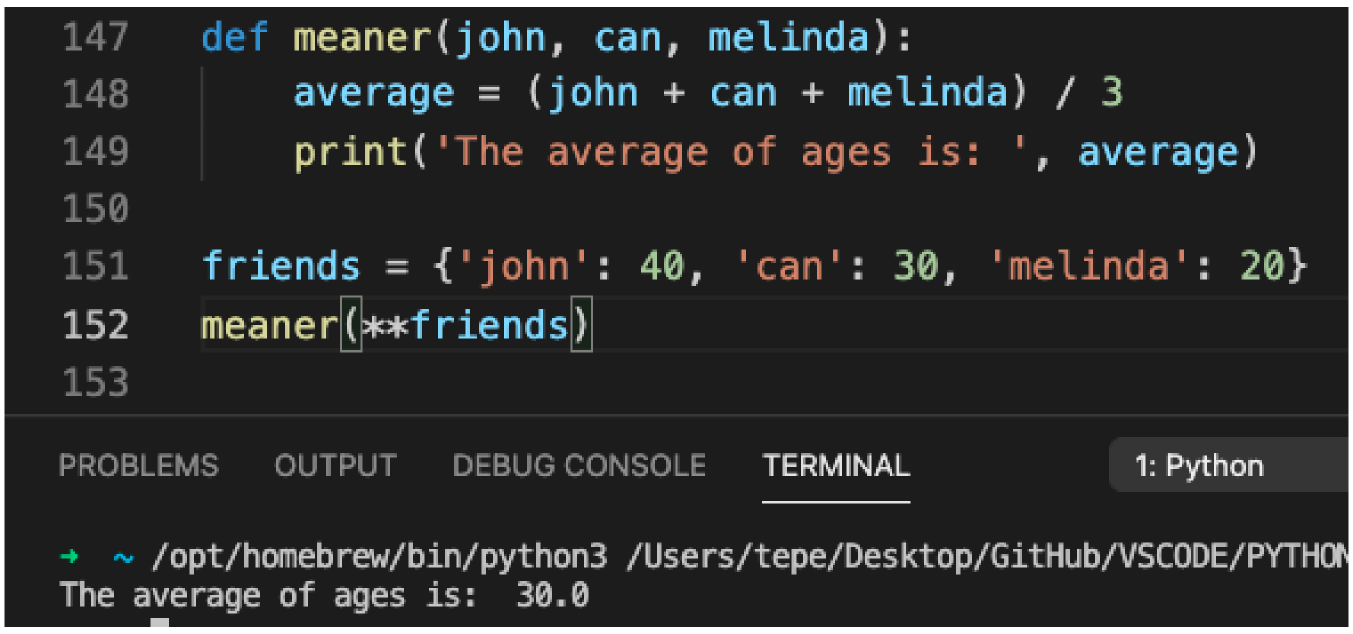
```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYT
ali is 33 years old.
sam is 45 years old.
john is 19 years old.
emily is 36 years old.
```

```
def name(multiple parameters)

    name(**variable)
```



```python
def gene(x = 'Solomon', y= 'David'):
    print(x, "belongs to Generation X")
    print(y, "belongs to Generation Y")
dict_gene = {'y' : "Marry", 'x' : "Fred"}
gene(**dict_gene)
print('--------')
gene()
```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSC
Fred belongs to Generation X
Marry belongs to Generation Y
--------
Solomon belongs to Generation X
David belongs to Generation Y
```



```python
def meaner(john, can, melinda):
    average = (john + can + melinda) / 3
    print('The average of ages is: ', average)

friends = {'john': 40, 'can': 30, 'melinda': 20}
meaner(**friends)

```

```
➜  ~ /opt/homebrew/bin/python3 /Users/tepe/Desktop/GitHub/VSCODE/PYTHON
The average of ages is:  30.0
```