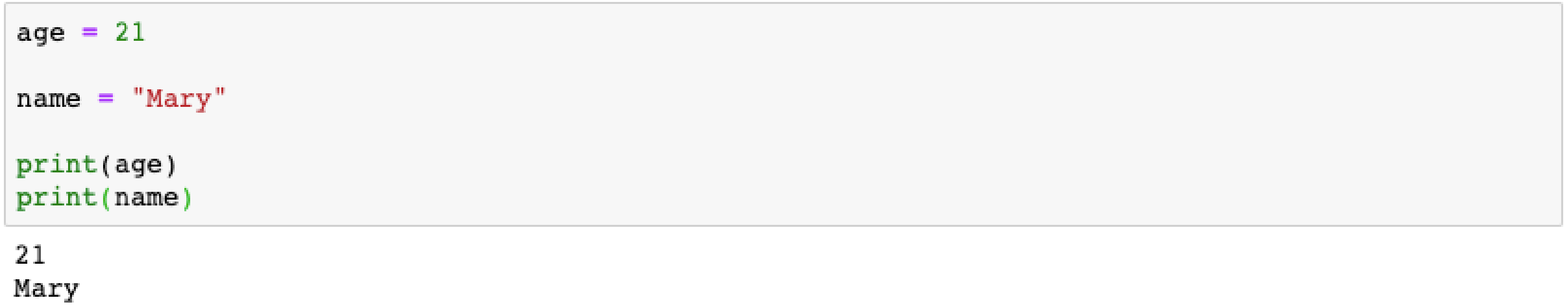
Introduction to Python

Agenda

- Variables
- Data Types
    - Fundamental Data Types
- Functions in Python – print()
- Operators
- Python Flow Control
- Pseudocodes

Variables

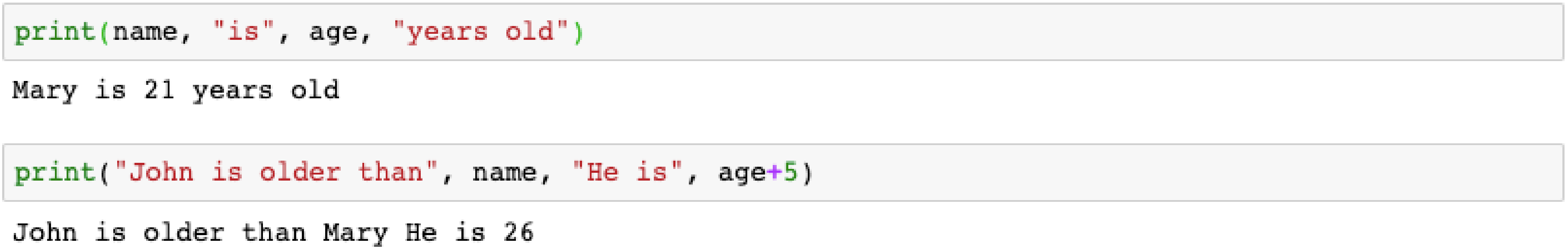
- Variables are containers that store data. Python has no command for declaring variable
- You simply assign a value to a variable to create it
- You use = to assign a value to a variable



```
age = 21

name = "Mary"

print(age)
print(name)
```

```
21
Mary
```

Print variables with texts



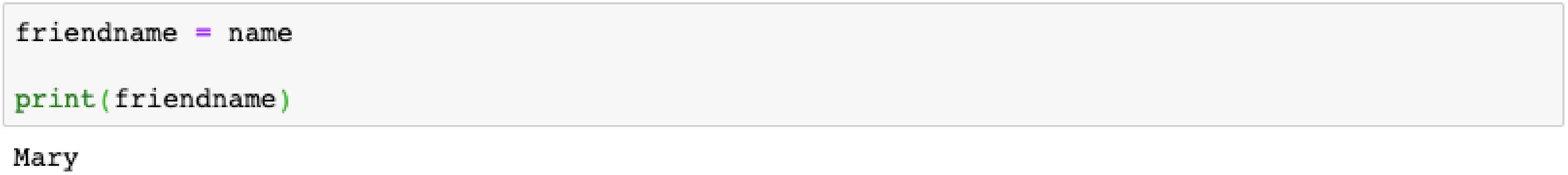
```
print(name, "is", age, "years old")
```

```
Mary is 21 years old
```

```
print("John is older than", name, "He is", age+5)
```

```
John is older than Mary He is 26
```

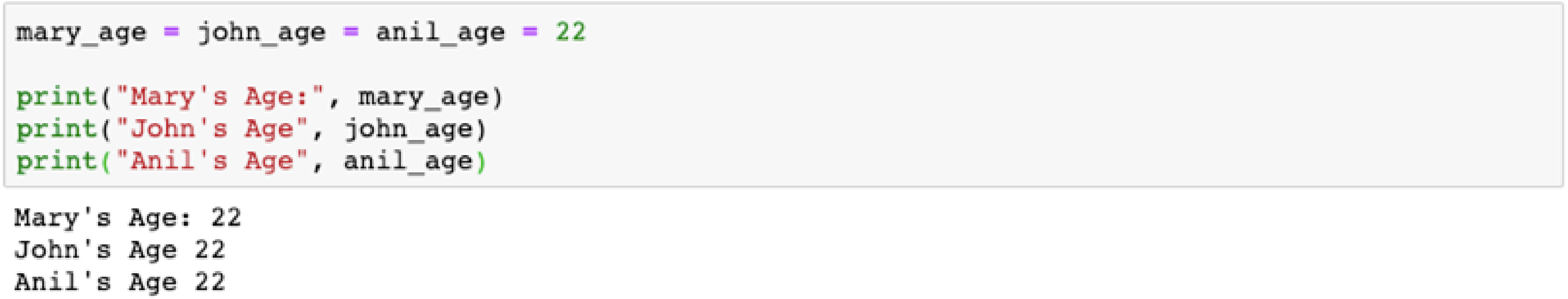
Reassign variable



```python
friendname = name

print(friendname)
```

```
Mary
```

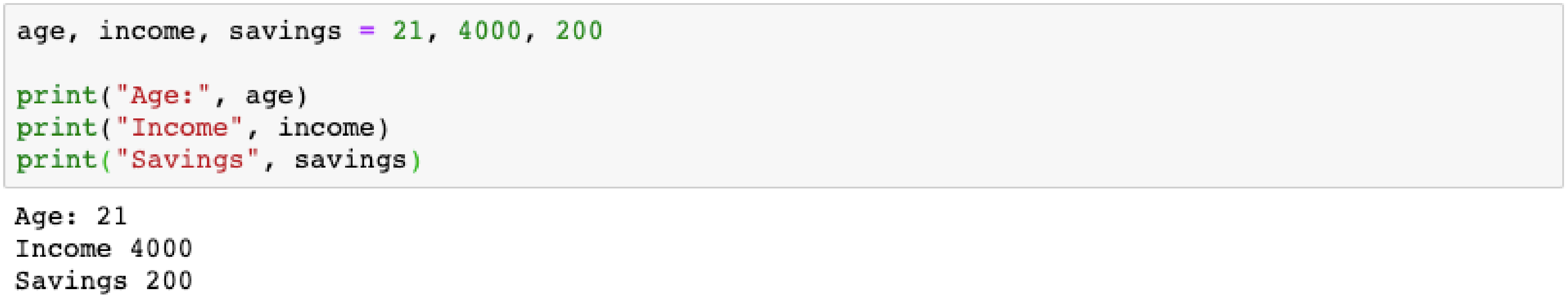
Multiple variables assignment with same value



```python
mary_age = john_age = anil_age = 22

print("Mary's Age:", mary_age)
print("John's Age", john_age)
print("Anil's Age", anil_age)
```

```
Mary's Age: 22
John's Age 22
Anil's Age 22
```

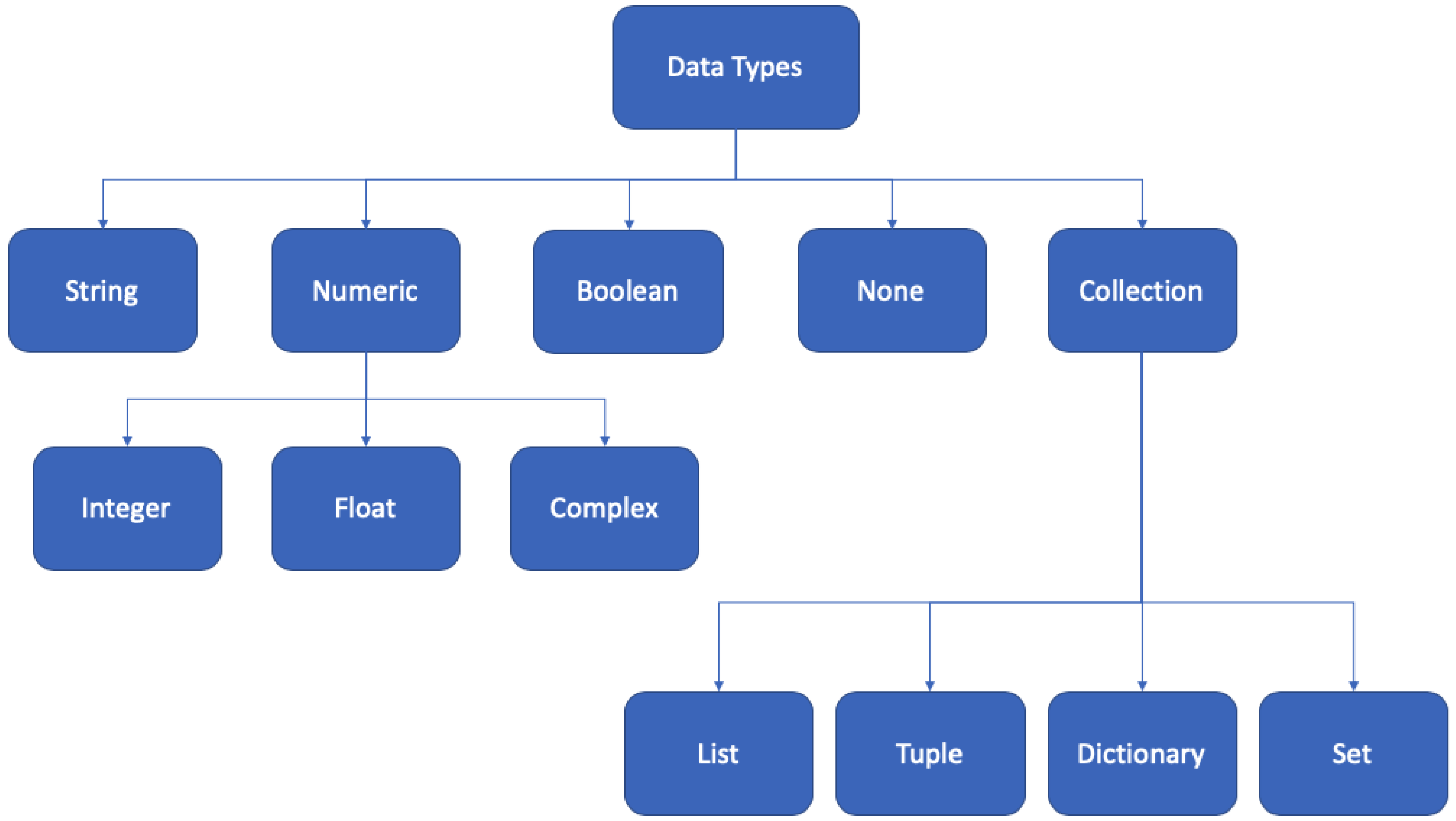
Multiple variables assignment with different values



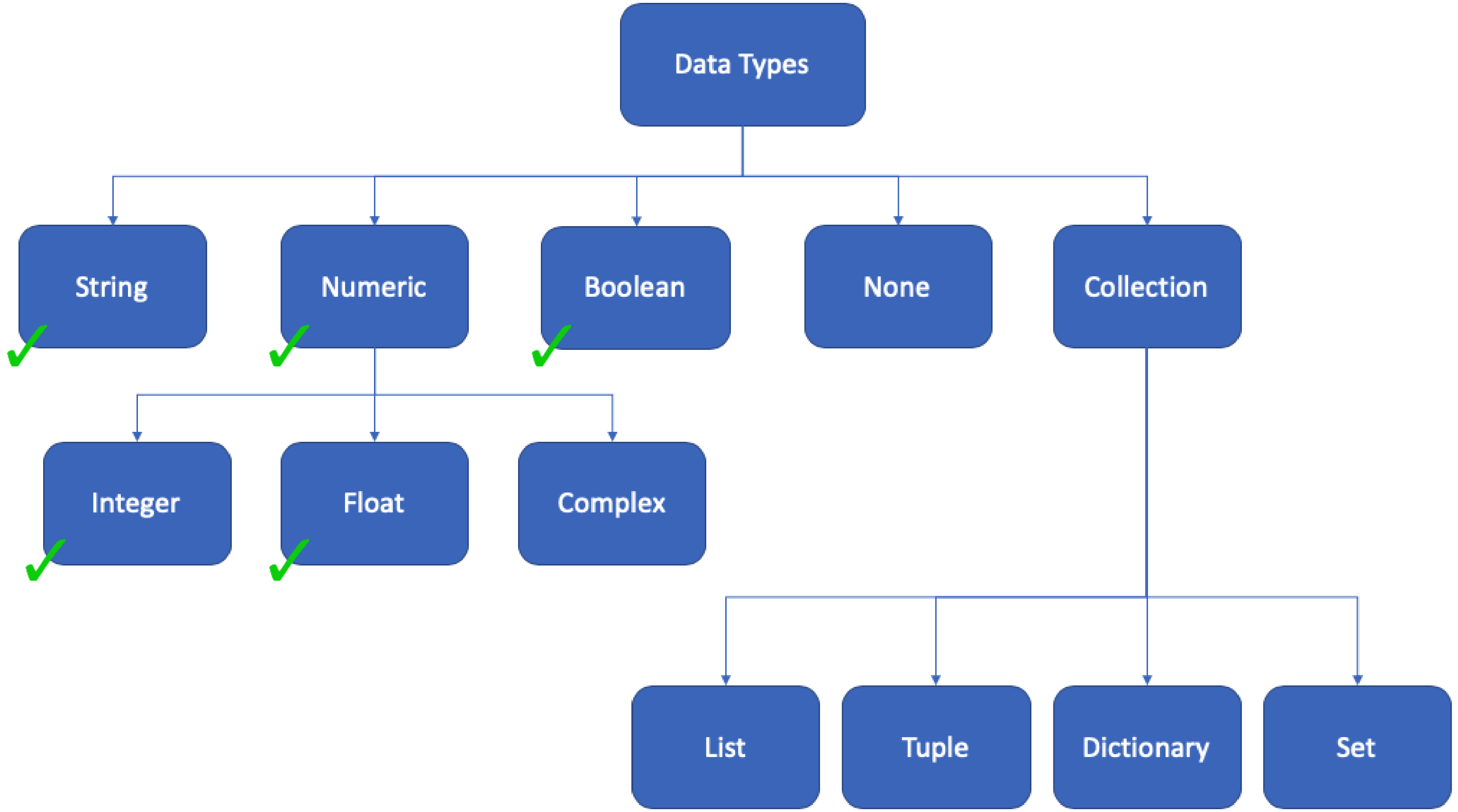
```python
age, income, savings = 21, 4000, 200

print("Age:", age)
print("Income", income)
print("Savings", savings)
```

```
Age: 21
Income 4000
Savings 200
```

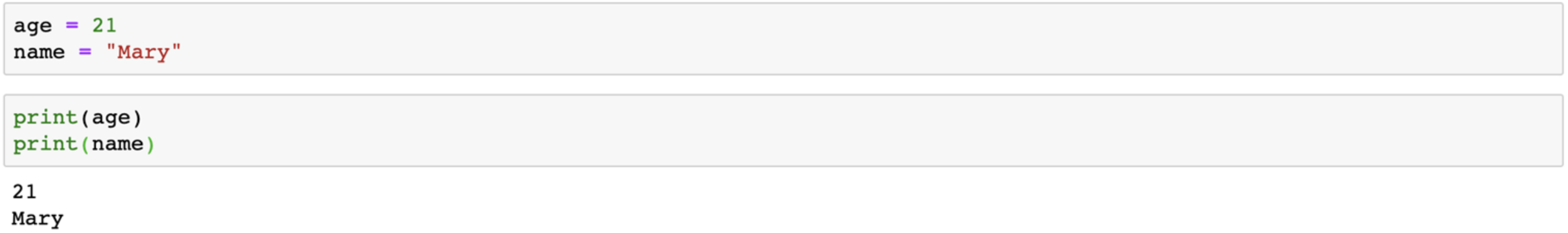
Python data types



We learn the following in our session today



Create integer & string variables

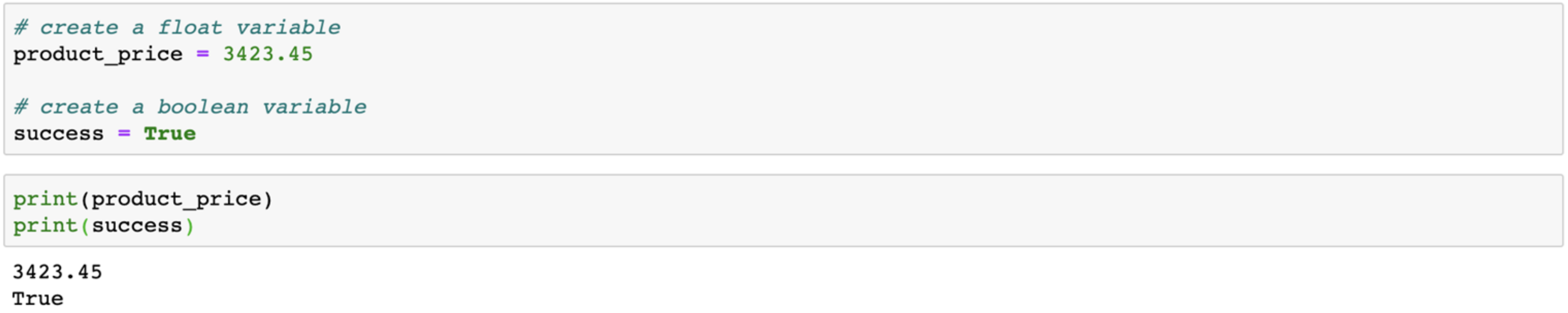


```
age = 21
name = "Mary"
```

```
print(age)
print(name)
```

```
21
Mary
```

Create float & boolean variables

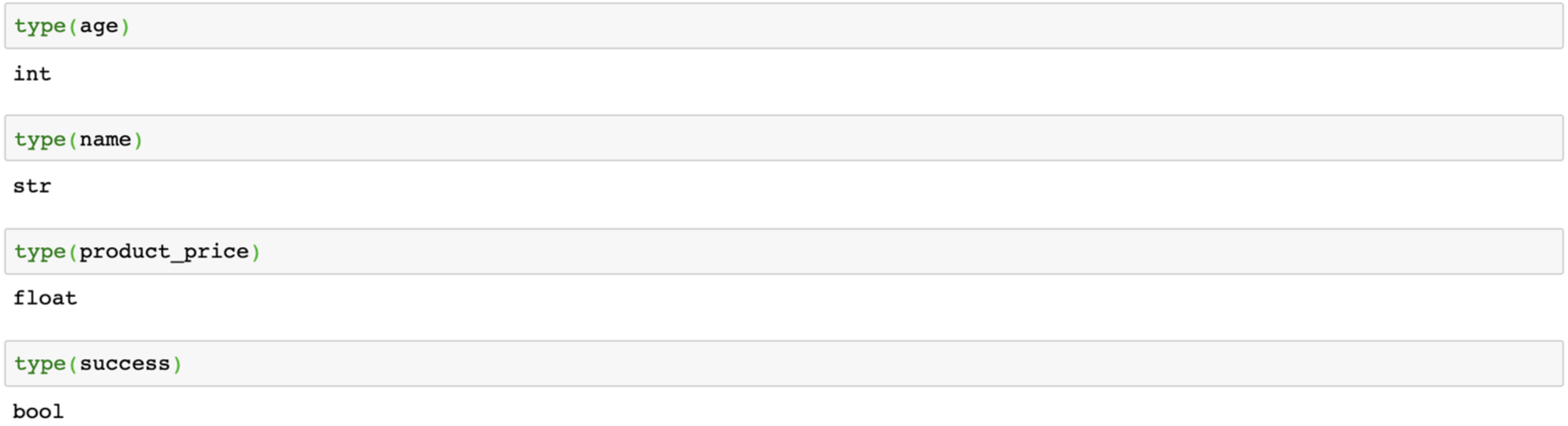


```
# create a float variable
product_price = 3423.45

# create a boolean variable
success = True
```

```
print(product_price)
print(success)
```

```
3423.45
True
```

Checking data types with type() function



```
type(age)
```

```
int
```

```
type(name)
```

```
str
```

```
type(product_price)
```

```
float
```

```
type(success)
```

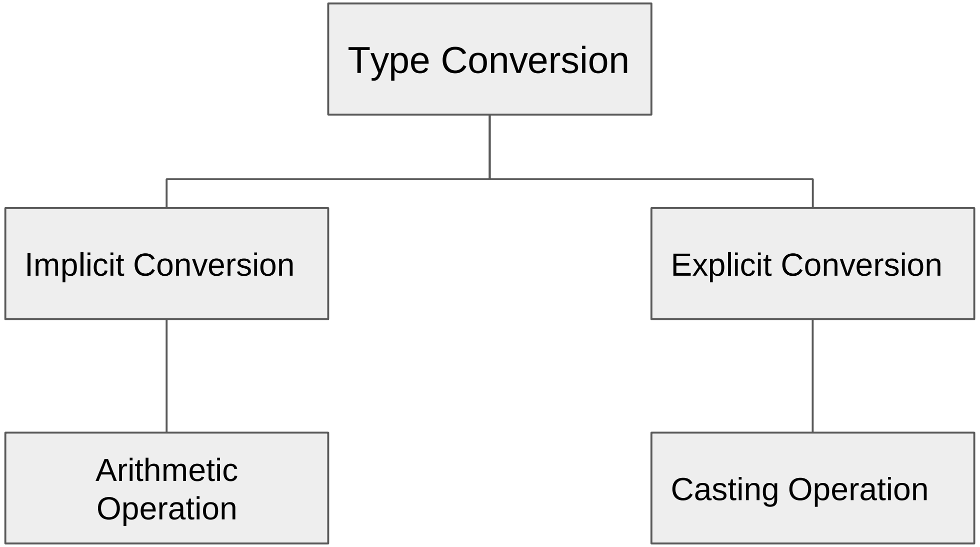
```
bool
```

Data type conversion

Python allows to convert one data type to another

Implicit Conversion: Conversion done by Python interpreter with programmer’s intervention

Explicit Conversion: Conversion that is user-defined that forces an expression to be of specific data type

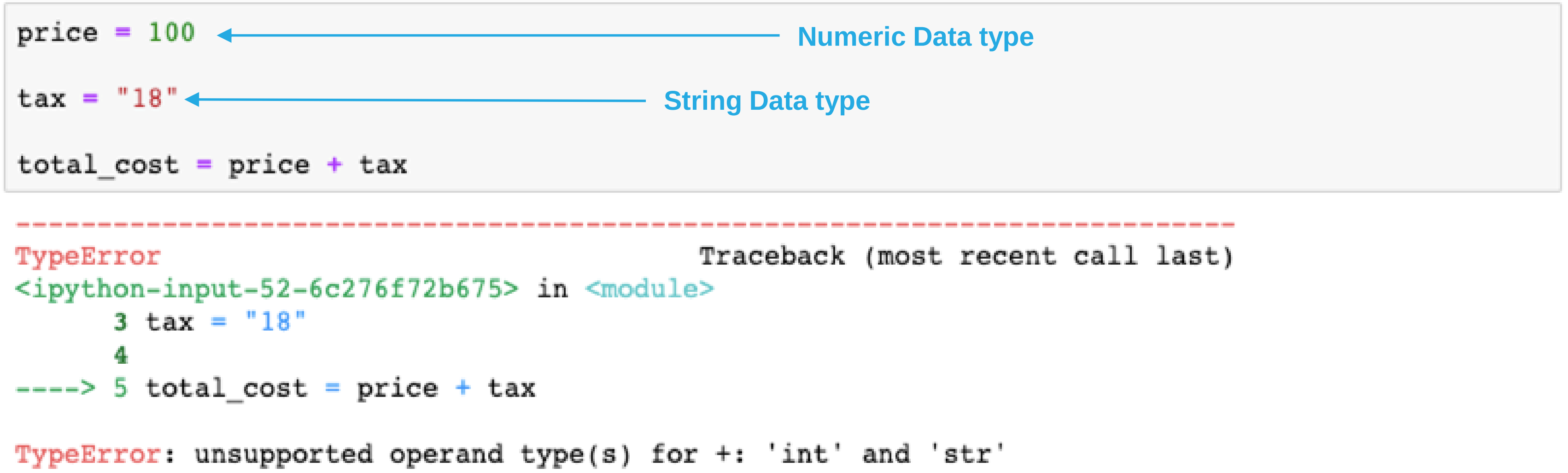


Implicit conversion

```
# Integer type data
income = 200

# Float type data
additional_income = 50.75

# Implicit conversion. Upon addition the resultant variable becomes float type
total_income = income + additional_income

print(total_income)
print(type(total_income))
```

```
250.75
<class 'float'>
```

Explicit conversion is required when you have mixed data types.



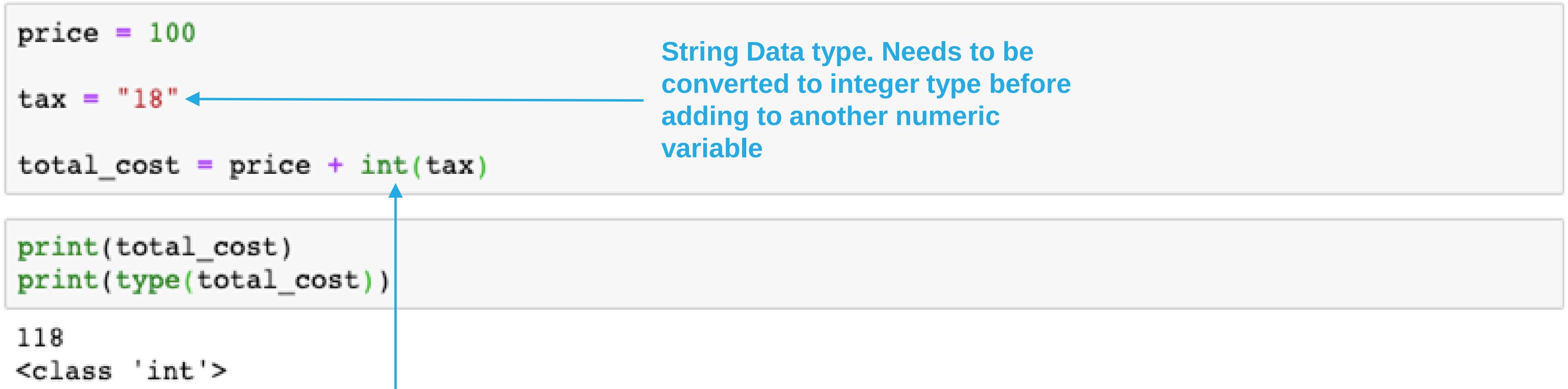
```
price = 100
tax = "18"
total_cost = price + tax
```

Numeric Data type

String Data type

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-52-6c276f72b675> in <module>
      3 tax = "18"
      4
----> 5 total_cost = price + tax

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

Explicit conversion



```
price = 100

tax = "18"

total_cost = price + int(tax)
```

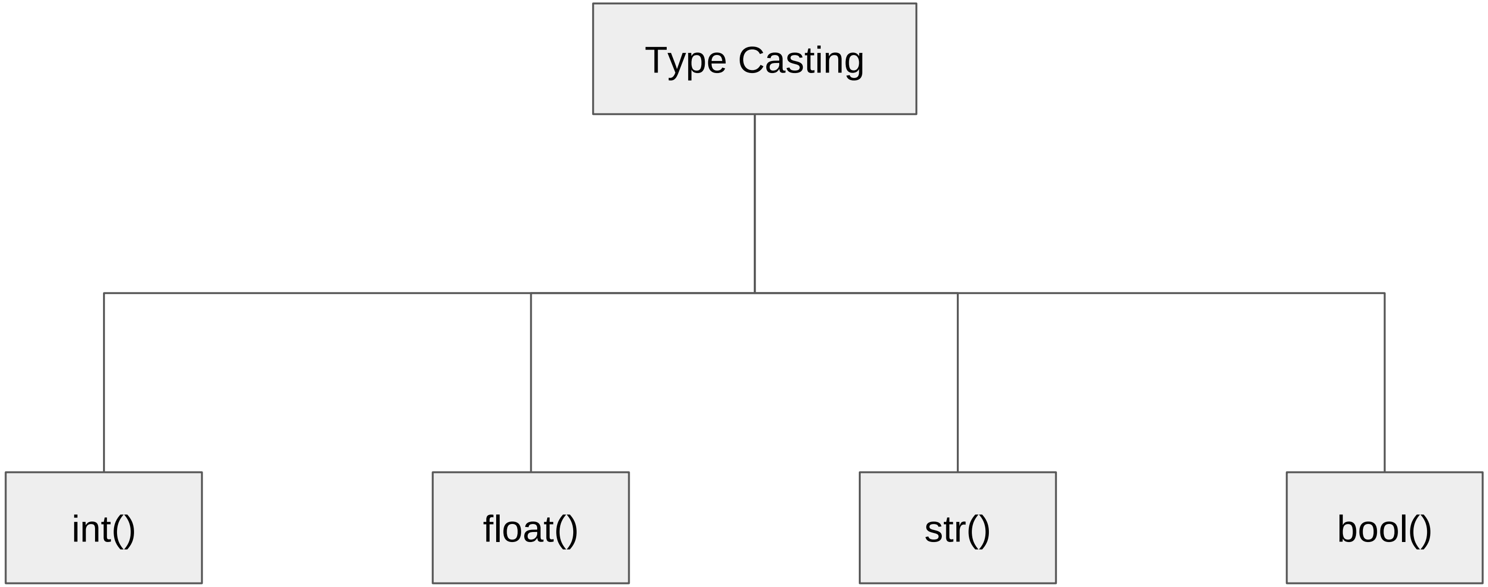
```
print(total_cost)
print(type(total_cost))
```

```
118
<class 'int'>
```

String Data type. Needs to be converted to integer type before adding to another numeric variable

Explicit type conversion of string to integer type

Type casting



Type casting

```
income = 10000.45
print(int(income))

age = 25
print(float(age))
```

```
10000
25.0
```

```
price = 34.6
str(price)
```

```
'34.6'
```

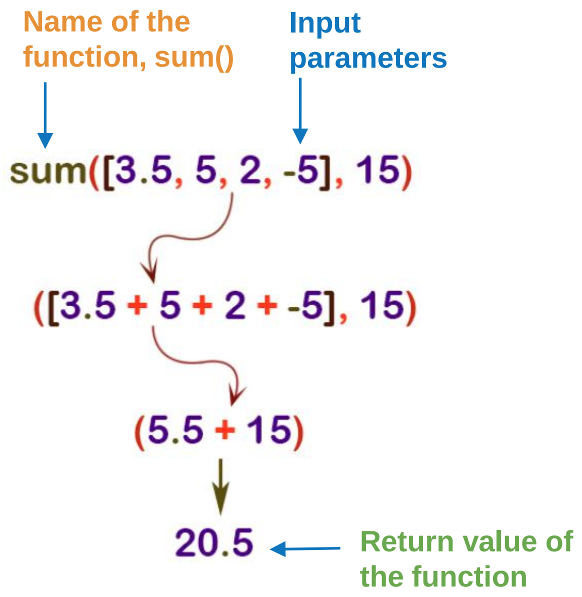
```
x, y = 0, 10
print(bool(x))
print(bool(y))
```

```
False
True
```

Functions in Python

What is a function?

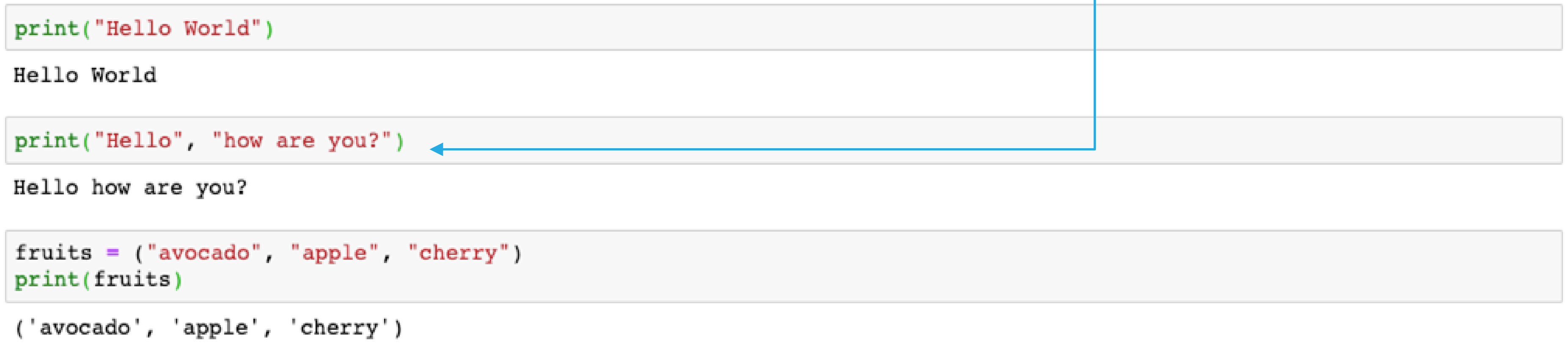
- A function is a block of code that runs when called
- A function has a name. You call the function by its name
- A function can take input(s), known as input parameters
- A function can return data as a result



The print() Function

The print() in python

Words or sentences separated by a comma within a print() function get concatenated when printed.



```python
print("Hello World")
```

```
Hello World
```

```python
print("Hello", "how are you?")
```

```
Hello how are you?
```

```python
fruits = ("avocado", "apple", "cherry")
print(fruits)
```

```
('avocado', 'apple', 'cherry')
```

The print() in python

The sep is an optional parameter. When output is printed, each word is separated by ^^^ characters

```
print("Hello", "how are you?", sep=" ^^^ ")
```

```
Hello ^^^ how are you?
```

```
print("Hello World")
print("\n")
print("How are you?")
```

```
Hello World



How are you?
```

print('\n') gives a new blank line

The backslash "\" is known as escape character. It is used in representing certain whitespaces.

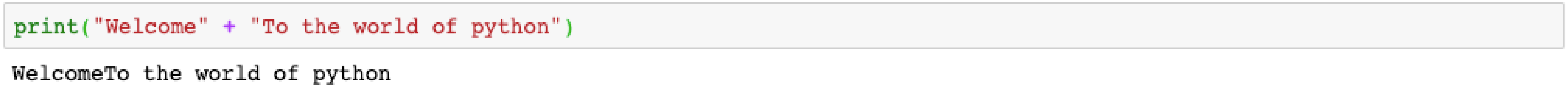
For example '\t' is a tab and '\n' is a new line.

Invalid use of opening and closing quotes

```
print('Welcome, Lets learn python")
```

```
  File "<ipython-input-17-3d5d6be15599>", line 1
    print('Welcome, Lets learn python")
                                       ^
SyntaxError: EOL while scanning string literal
```

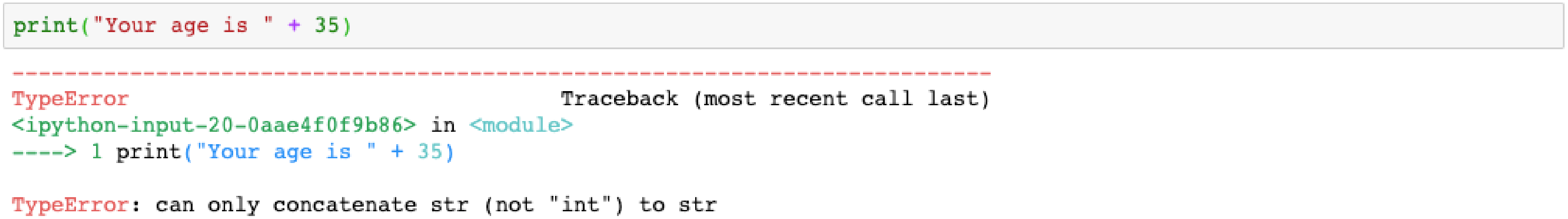
Print with concatenation



```python
print("Welcome" + "To the world of python")
```

```
WelcomeTo the world of python
```

You cannot concatenate string & number

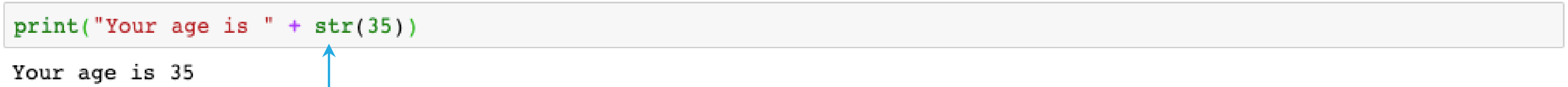


```
print("Your age is " + 35)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-0aae4f0f9b86> in <module>
----> 1 print("Your age is " + 35)

TypeError: can only concatenate str (not "int") to str
```

That’s wrong because adding numbers to strings doesn’t make any sense. You need to explicitly convert the number to string first, in order to join them together.

Concatenating with type casting



```
print("Your age is " + str(35))
```

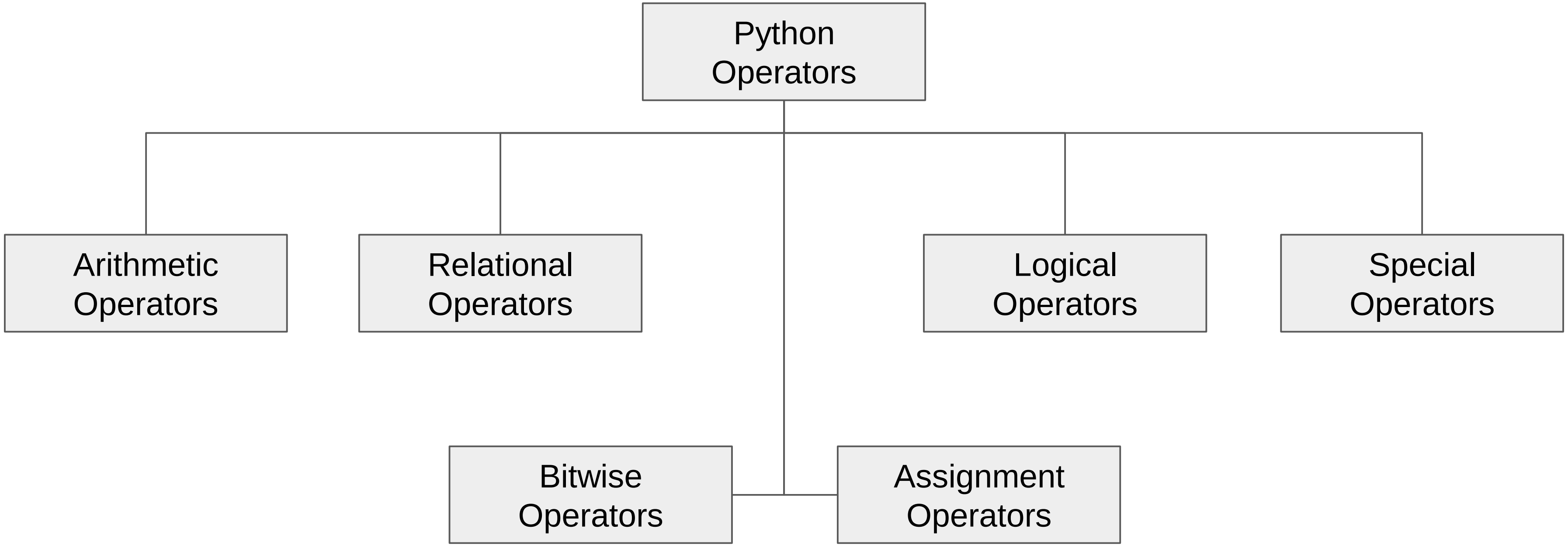
```
Your age is 35
```

Explicit conversion of a number to string with str() function.

We learn more such type conversions in our upcoming sessions

Python Operators

Python operators



Arithmetic operators: Addition

```
# Most common Arithmetic Operators are:
        + (Addition)
        - (Subtraction)
        * (Multiplication)
        / (Division)
        % (Modulus)
        ** (Square)
```

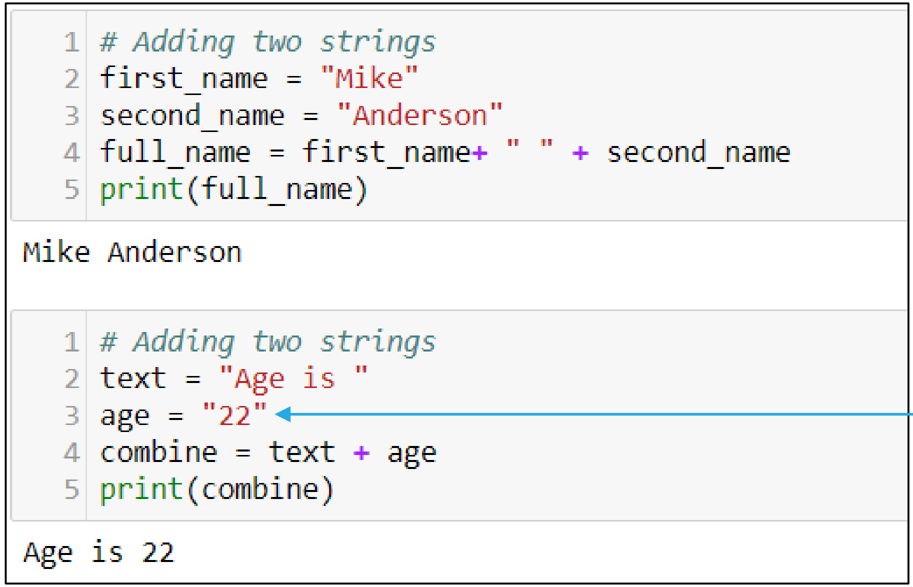
```
# Adding 2 integer variables
price = 100
tax = 25
total_cost = price + tax
print(total_cost)
```

```
125
```

```
# Adding1 integer variable and 1 float variable
price = 100
tax = 12.80
total_cost = price + tax
print(total_cost)
```

```
112.8
```

Arithmetic operators: Addition



```
# Adding two strings
first_name = "Mike"
second_name = "Anderson"
full_name = first_name+ " " + second_name
print(full_name)
```

```
Mike Anderson
```

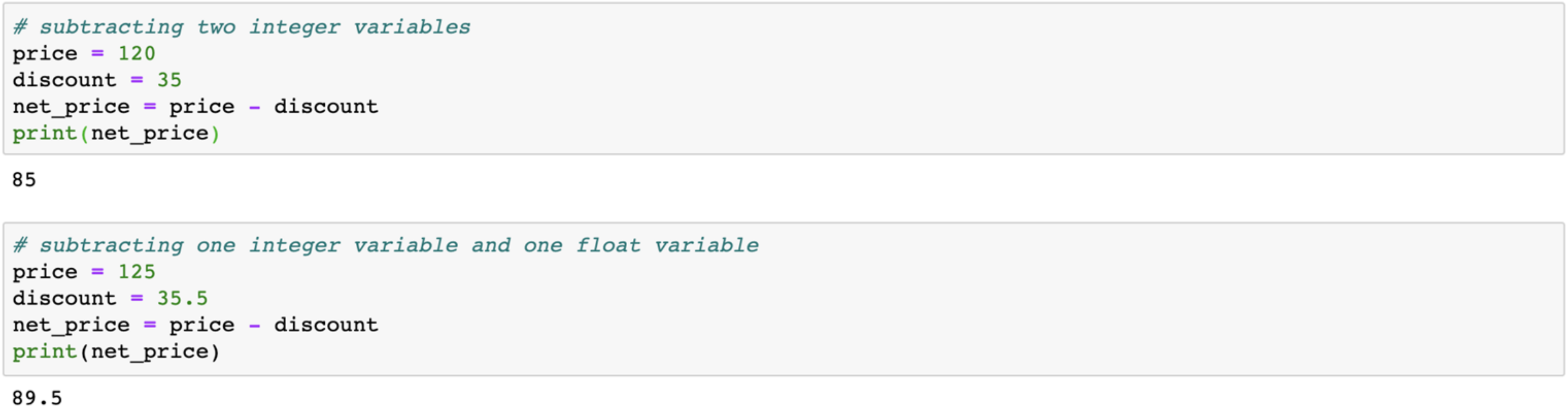
```
# Adding two strings
text = "Age is "
age = "22"
combine = text + age
print(combine)
```

```
Age is 22
```

Note that 22 is a string here because it has been put within the quotes.

Here two strings are getting concatenated.

Arithmetic operators: Subtraction



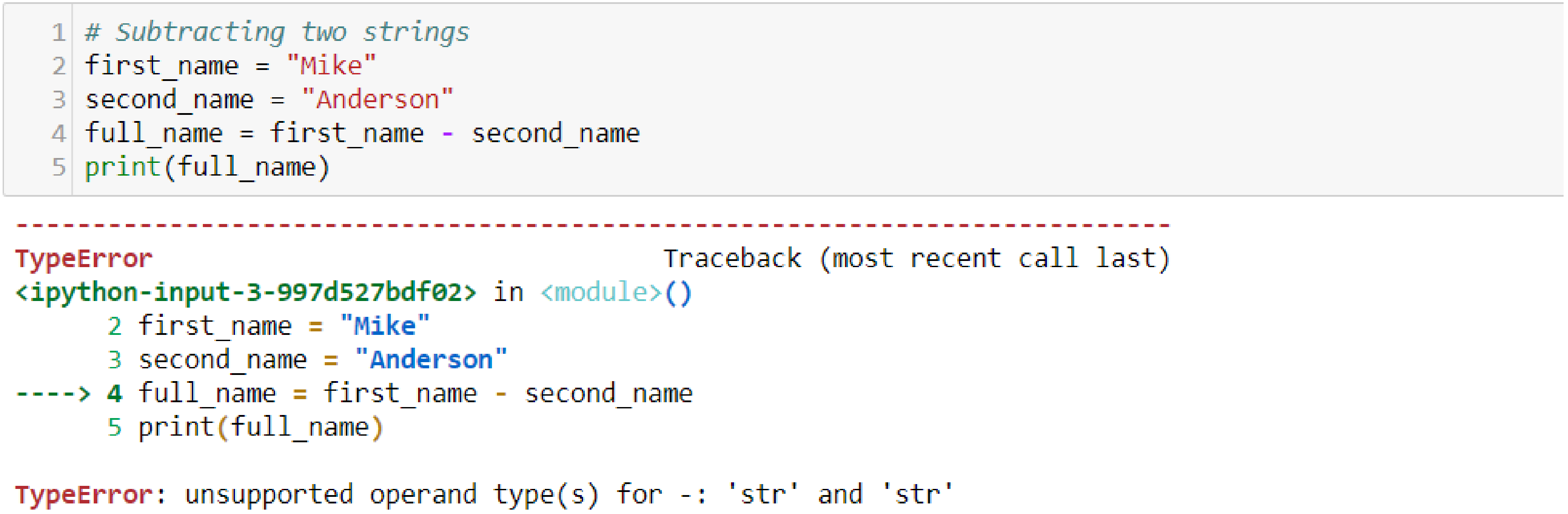
```
# subtracting two integer variables
price = 120
discount = 35
net_price = price - discount
print(net_price)
```

```
85
```

```
# subtracting one integer variable and one float variable
price = 125
discount = 35.5
net_price = price - discount
print(net_price)
```

```
89.5
```

Arithmetic operators: Subtraction



```python
1 # Subtracting two strings
2 first_name = "Mike"
3 second_name = "Anderson"
4 full_name = first_name - second_name
5 print(full_name)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-997d527bdf02> in <module>()
      2 first_name = "Mike"
      3 second_name = "Anderson"
----> 4 full_name = first_name - second_name
      5 print(full_name)

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

When two strings are added, the operator basically concatenates the two strings. Subtracting two strings does not make any sense.

Arithmetic operators: Multiplication

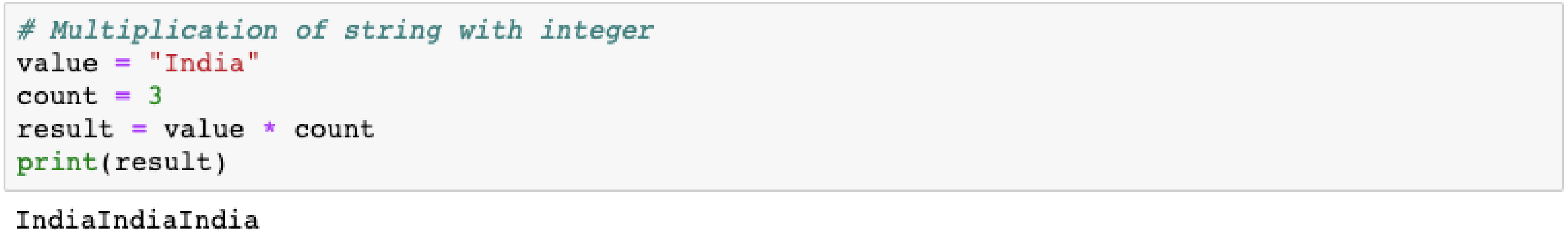
```
# Multiplication of 2 integer variables
price = 120
quantity = 15
total_cost = price * quantity
print(total_cost)
```

```
1800
```

```
# Multiplication of 1 integer & 1 float variable
price = 120.50
quantity = 15
total_cost = price * quantity
print(total_cost)
```

```
1807.5
```

Arithmetic operators: Multiplication



```
# Multiplication of string with integer
value = "India"
count = 3
result = value * count
print(result)
```

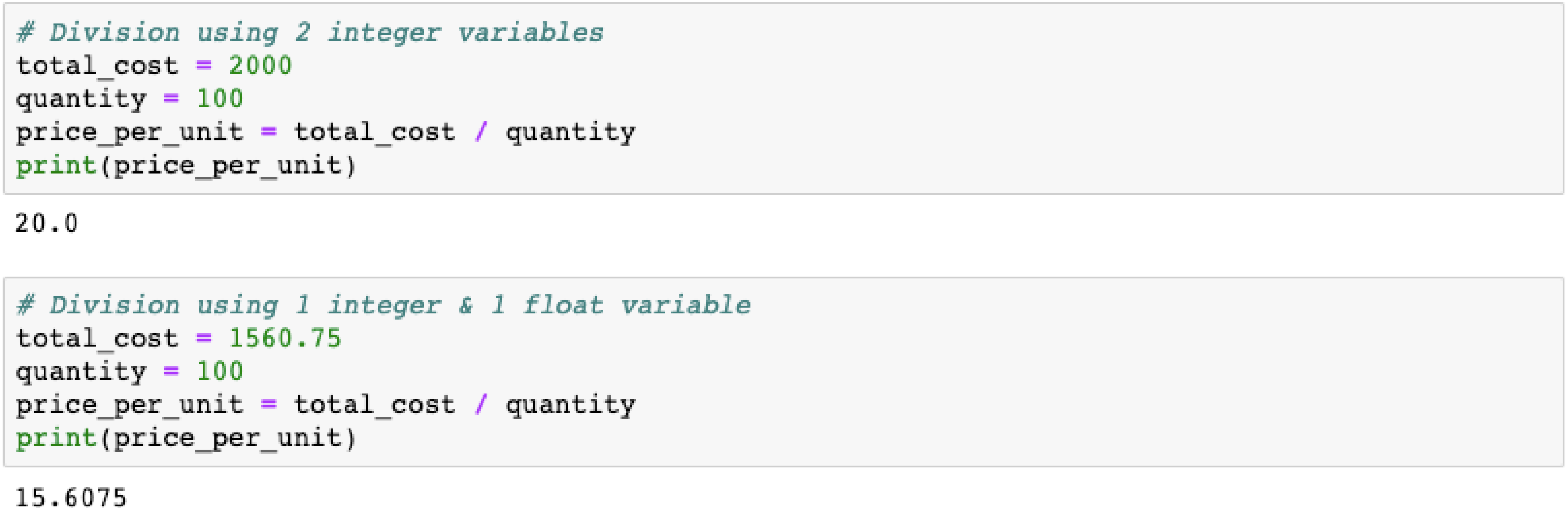
```
IndiaIndiaIndia
```

Arithmetic operators: Multiplication

```python
# Multiplication of 2 strings not possible
value = "Boat"
count = "3"
result = value * count
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-77-25575f6e63ae> in <module>
      2 value = "Boat"
      3 count = "3"
----> 4 result = value * count
      5 print(result)

TypeError: can't multiply sequence by non-int of type 'str'
```

Arithmetic operators: Division



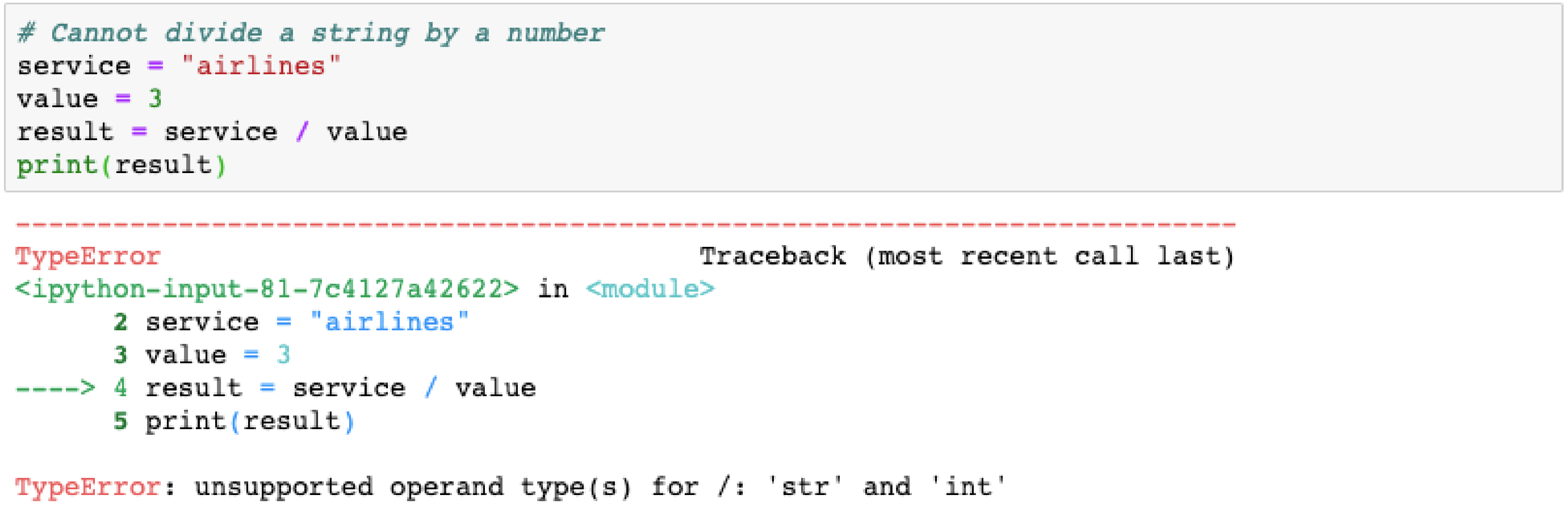
```
# Division using 2 integer variables
total_cost = 2000
quantity = 100
price_per_unit = total_cost / quantity
print(price_per_unit)
```

```
20.0
```

```
# Division using 1 integer & 1 float variable
total_cost = 1560.75
quantity = 100
price_per_unit = total_cost / quantity
print(price_per_unit)
```

```
15.6075
```

Arithmetic operators: Division



```
# Cannot divide a string by a number
service = "airlines"
value = 3
result = service / value
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-7c4127a42622> in <module>
      2 service = "airlines"
      3 value = 3
----> 4 result = service / value
      5 print(result)

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

Arithmetic operators: Get quotient

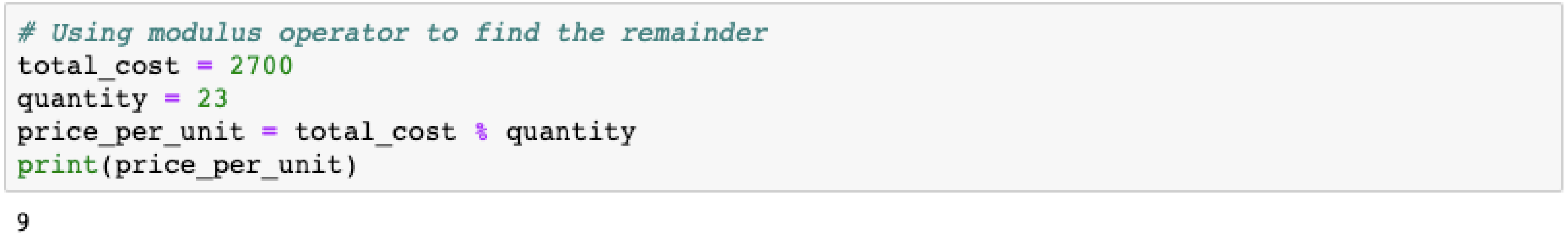
```
# division operation to get quotient
# it is known as floor division
total_cost = 13000
quantity = 23
price_per_unit = total_cost // quantity
print(price_per_unit)
```

```
565
```

```
# division operation to get quotient
# it is known as floor division
total_cost = 13000.50
quantity = 23
price_per_unit = total_cost // quantity
print(price_per_unit)
```

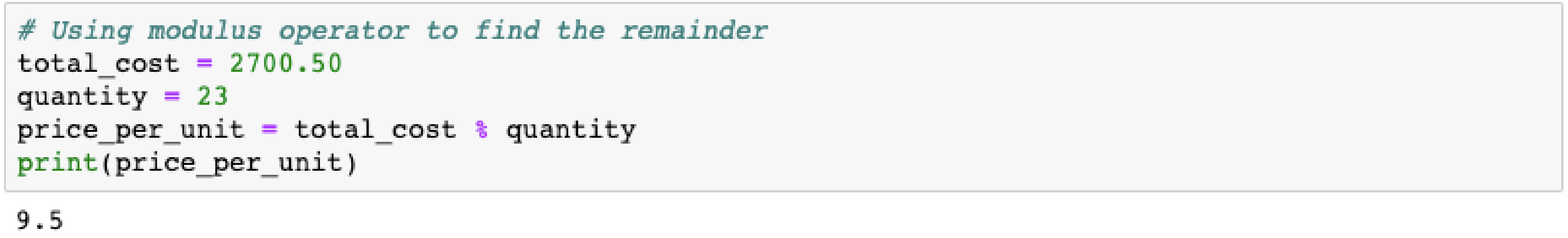
```
565.0
```

Arithmetic operators: Get remainder



```
# Using modulus operator to find the remainder
total_cost = 2700
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

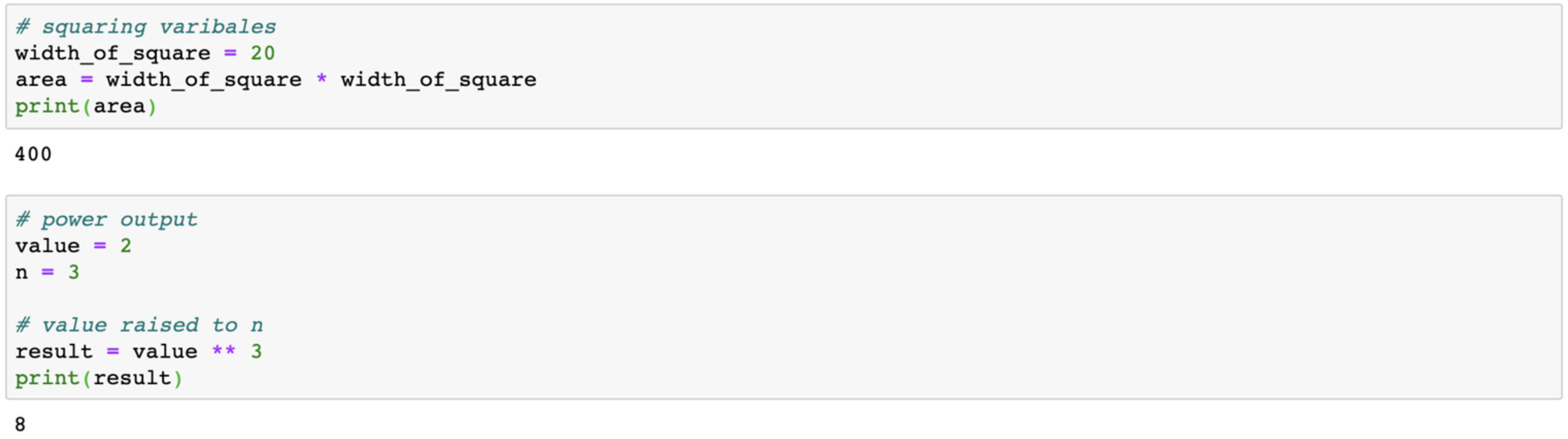
```
9
```



```
# Using modulus operator to find the remainder
total_cost = 2700.50
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

```
9.5
```

Arithmetic operators: Get square & power output



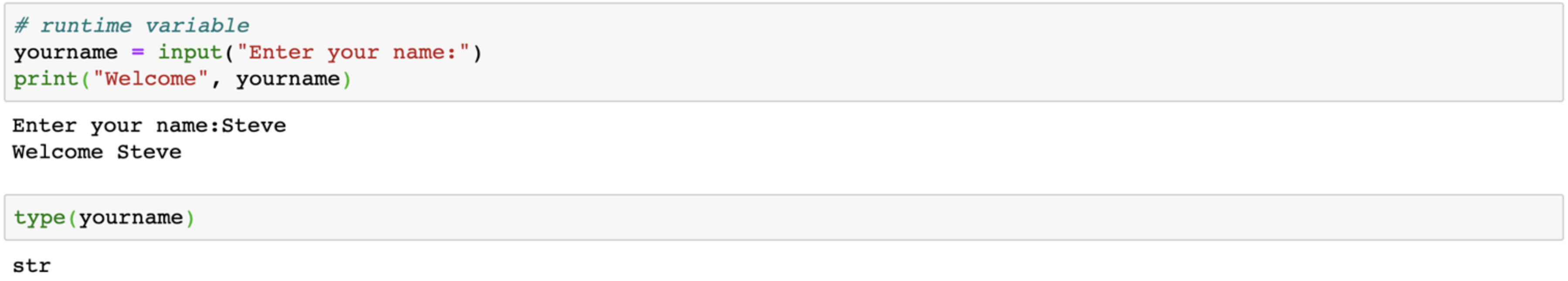
```
# squaring varibales
width_of_square = 20
area = width_of_square * width_of_square
print(area)
```

```
400
```

```
# power output
value = 2
n = 3

# value raised to n
result = value ** 3
print(result)
```

```
8
```

Runtime variable



```python
# runtime variable
yourname = input("Enter your name:")
print("Welcome", yourname)
```

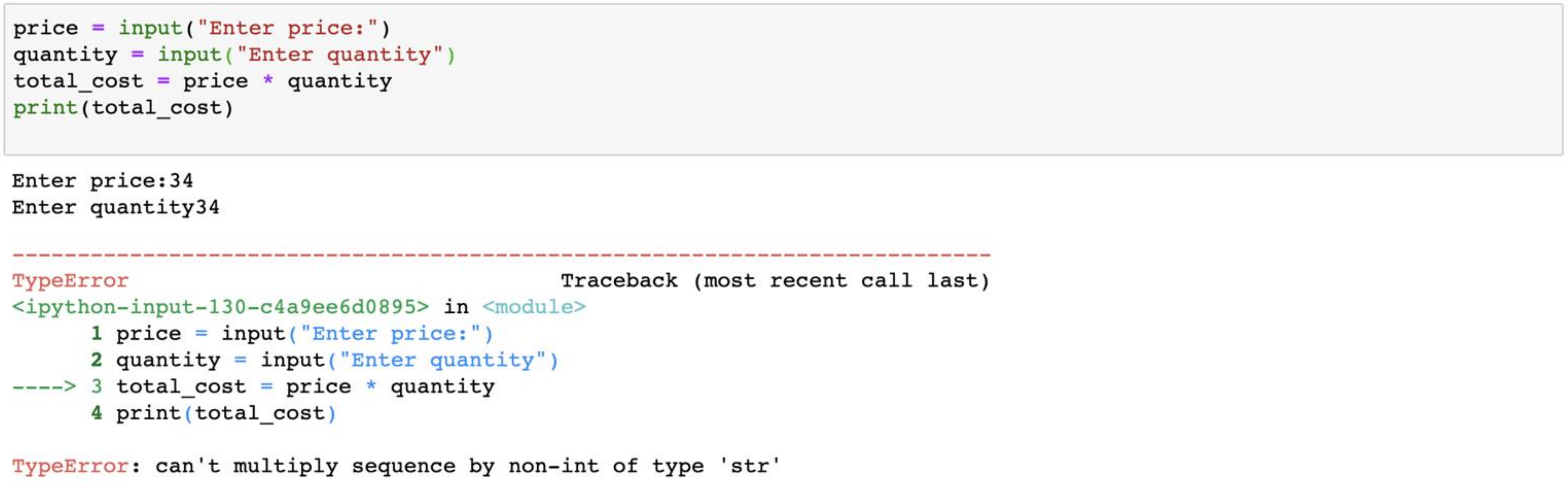
```
Enter your name:Steve
Welcome Steve
```

```python
type(yourname)
```

```
str
```

Runtime variable

The default return type of a runtime variable is string.

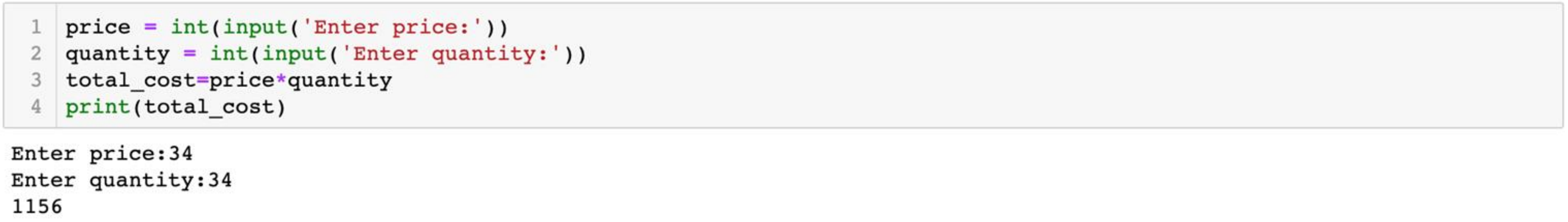


```
price = input("Enter price:")
quantity = input("Enter quantity")
total_cost = price * quantity
print(total_cost)
```

```
Enter price:34
Enter quantity34

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-130-c4a9ee6d0895> in <module>
      1 price = input("Enter price:")
      2 quantity = input("Enter quantity")
----> 3 total_cost = price * quantity
      4 print(total_cost)

TypeError: can't multiply sequence by non-int of type 'str'
```

Runtime variable

The default return type of a runtime variable is string. So in order to use the entered value for computation we convert the values to integer.



```
price = int(input('Enter price:'))
quantity = int(input('Enter quantity:'))
total_cost=price*quantity
print(total_cost)
```

```
Enter price:34
Enter quantity:34
1156
```

Relational operators

Relational operators are used to compare 2 values and take certain decisions based on the outcome (outcome is is a boolean value)

<	is less than
<=	is less than & equal to
>	is greater than
>=	is greater than & equal to
==	is equal to
!=	is not equal to

Relational operators

```python
mona_age = 35
john_age = 25
```

```python
mona_age > john_age
```

```
True
```

```python
mona_age < john_age
```

```
False
```

```python
mona_age == john_age
```

```
False
```

```python
mona_age >= john_age
```

```
True
```

```python
mona_age <= john_age
```

```
False
```

```python
mona_age != john_age
```

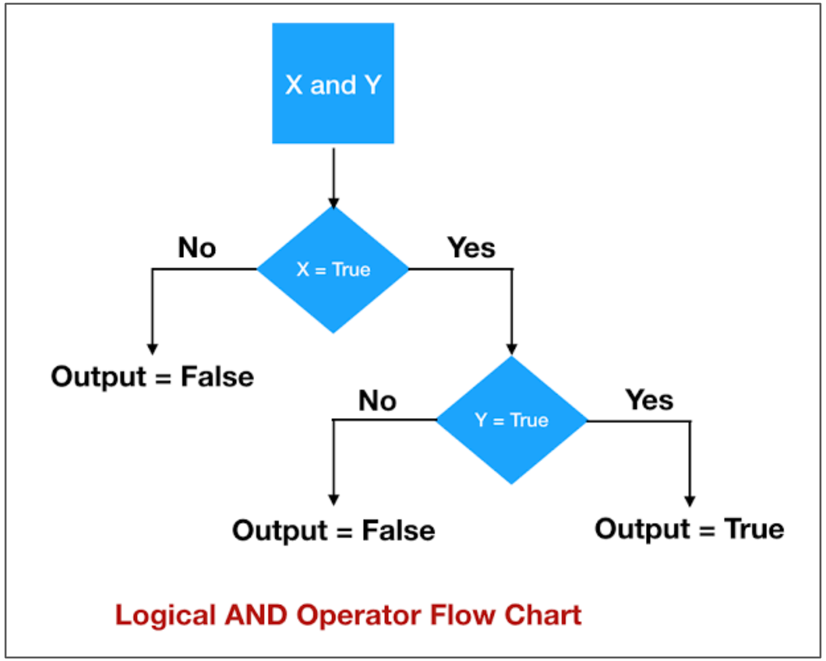
```
True
```

Logical operators

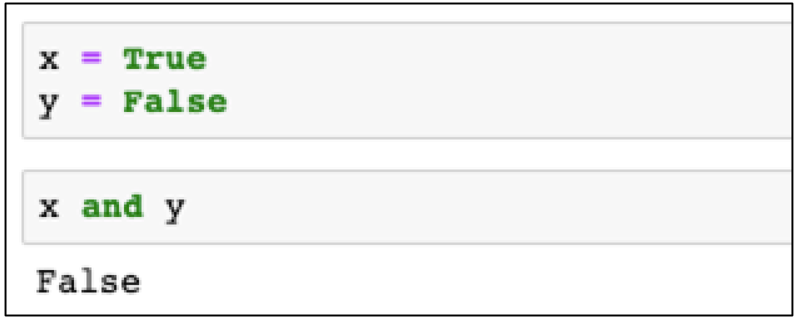
Logical operators in Python are used for conditional statements are either True or False

AND	Returns True if both the operands are True
OR	Returns True if either of the operands are True
NOT	Returns True if the operand is False

AND operator



Logical AND Operator Flow Chart

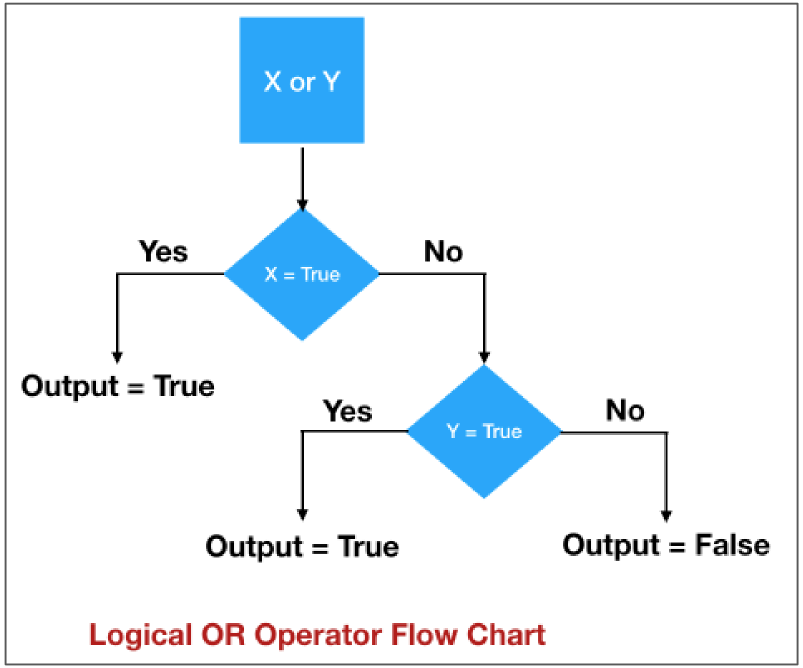


```
x = True
y = False
```

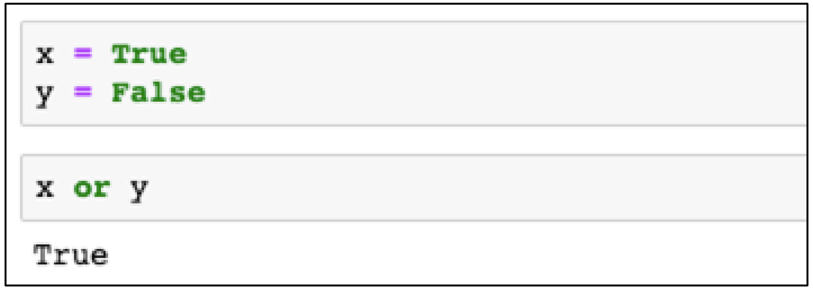
```
x and y
```

```
False
```

OR operator



Logical OR Operator Flow Chart

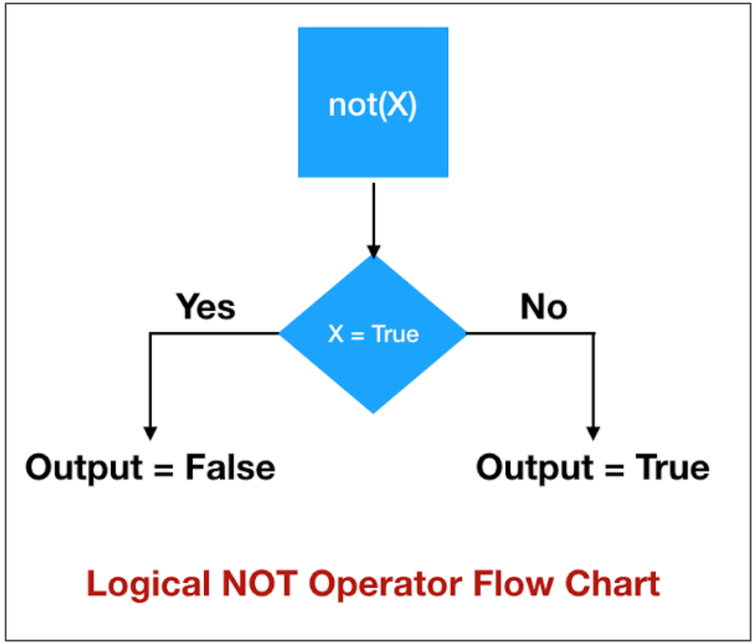


```
x = True
y = False
```

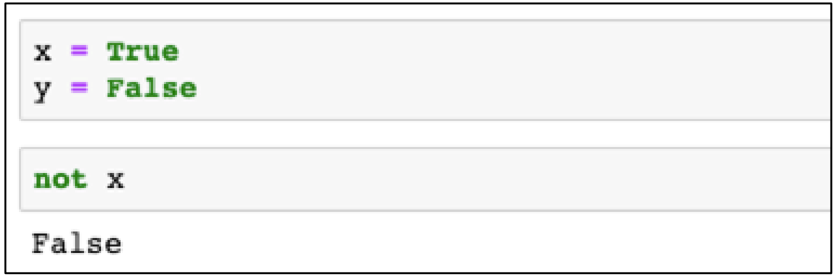
```
x or y
```

```
True
```

NOT operator



Logical NOT Operator Flow Chart



```
x = True
y = False
```

```
not x
```

```
False
```

Membership operators

Membership operators tests whether a value is a member of a sequence. The sequence may be a list, a string, a tuple, or a dictionary.

in

The ‘in’ operator is used to check if a value exists in any sequence object or not. For example, if 2 exists in a list, say [4, 5, 7, 2]. This evaluates to True if it finds a value in the specified sequence object. Otherwise it returns a False.

Membership operators

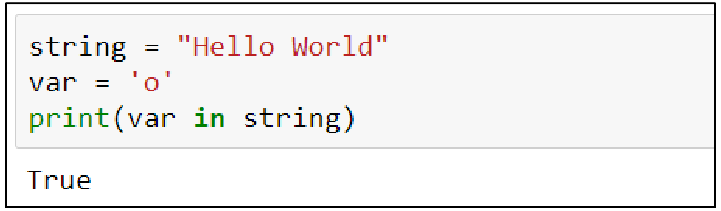
not in

A ‘not in’ works in an opposite way to an ‘in’ operator. A ‘not in’ evaluates to True if a value is not found in the specified sequence object. Else it returns a False.

Membership operators

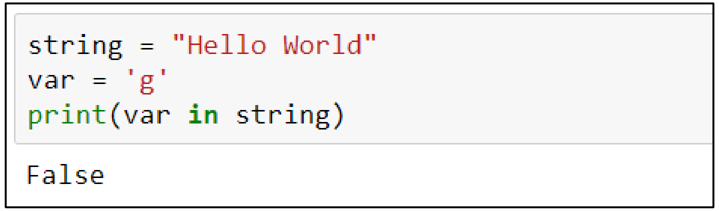
Example:

in operator:



```
string = "Hello World"
var = 'o'
print(var in string)
```

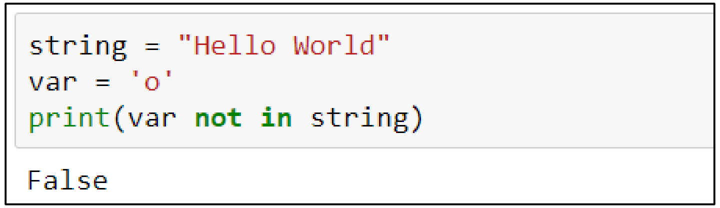
```
True
```



```
string = "Hello World"
var = 'g'
print(var in string)
```

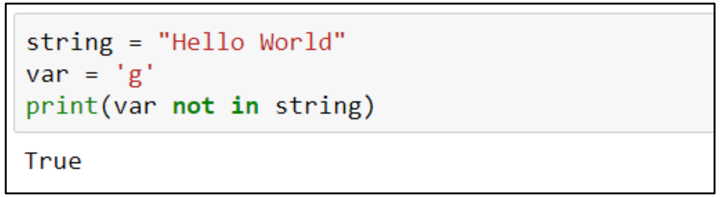
```
False
```

not in operator:



```
string = "Hello World"
var = 'o'
print(var not in string)
```

```
False
```



```
string = "Hello World"
var = 'g'
print(var not in string)
```

```
True
```

Some bitwise operators

Python Bitwise Operators take one to two operands, and operates on it/them bit by bit, instead of whole

& (Bitwise and)	The binary and (&) takes two values and performs an AND-ing on each pair of bits.
| (Bitwise or)	Compared to &, this one returns 1 even if one of the two corresponding bits from the two operands is 1.

The truth tables

&
(Bitwise and)

Value	Value	Value & Value
True	True	True
True	False	False
False	True	False
False	False	False

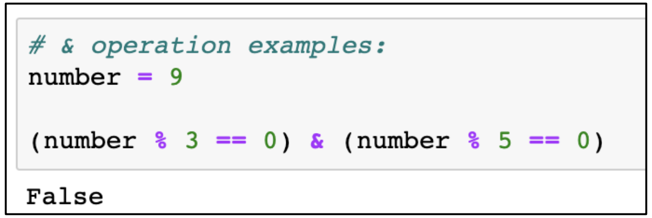
|
(Bitwise or)

Value	Value	Value | Value
True	True	True
True	False	True
False	True	True
False	False	False

Some bitwise operators

Example:

& operator:



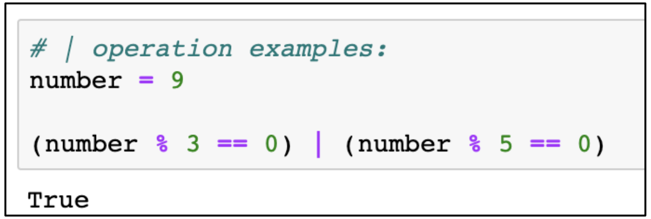
```
# & operation examples:
number = 9

(number % 3 == 0) & (number % 5 == 0)

False
```

(9 % 3 == 0) & (9 % 5 == 0)

Implies True & False

This results in Fasle.

| operator:



```
# | operation examples:
number = 9

(number % 3 == 0) | (number % 5 == 0)

True
```

(9 % 3 == 0) | (9 % 5 == 0)

Implies True | False

This results in True.

More assignment operators

The assignment operators are used to store data into a variable.

a += b	is same as a =	a + b
a *= b	is same as a =	a * b
a /= b	is same as a =	a / b
a %= b	is same as a =	a % b
a **= b	is same as a =	a ** b
a //= b	is same as a =	a // b

Data slicing

The basic syntax for a slice is square brackets with colons and integers inside "[0:1:2]".

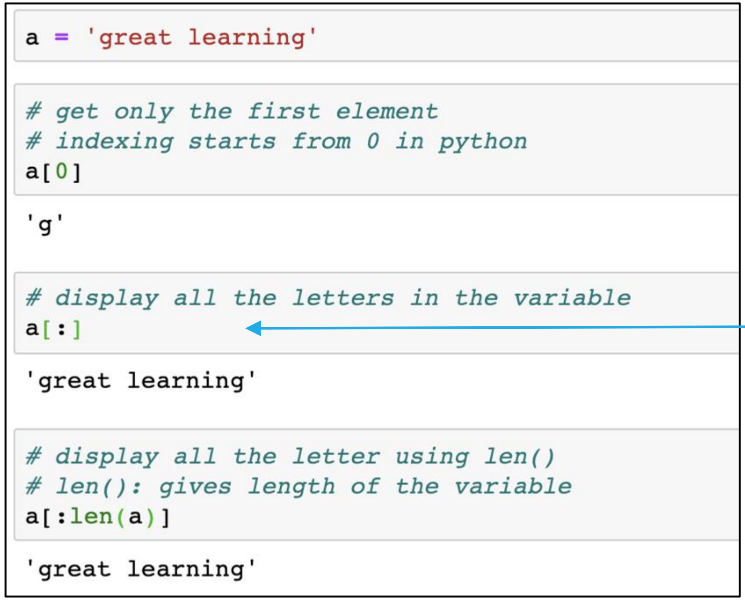
myStr[start : stop : step]

myStr[: stop] # By using one colon and leaving the first argument blank we automatically start at index 0, stepping by 1

myStr[start :] # By using one colon and leaving the last argument blank we automatically go to the end, stepping by 1

myStr[: : step] # by using two colons and leaving the first two arguments blank we start at index 0, go to the end and step by 1

Data slicing



```
a = 'great learning'
```

```
# get only the first element
# indexing starts from 0 in python
a[0]
```

```
'g'
```

```
# display all the letters in the variable
a[:]
```

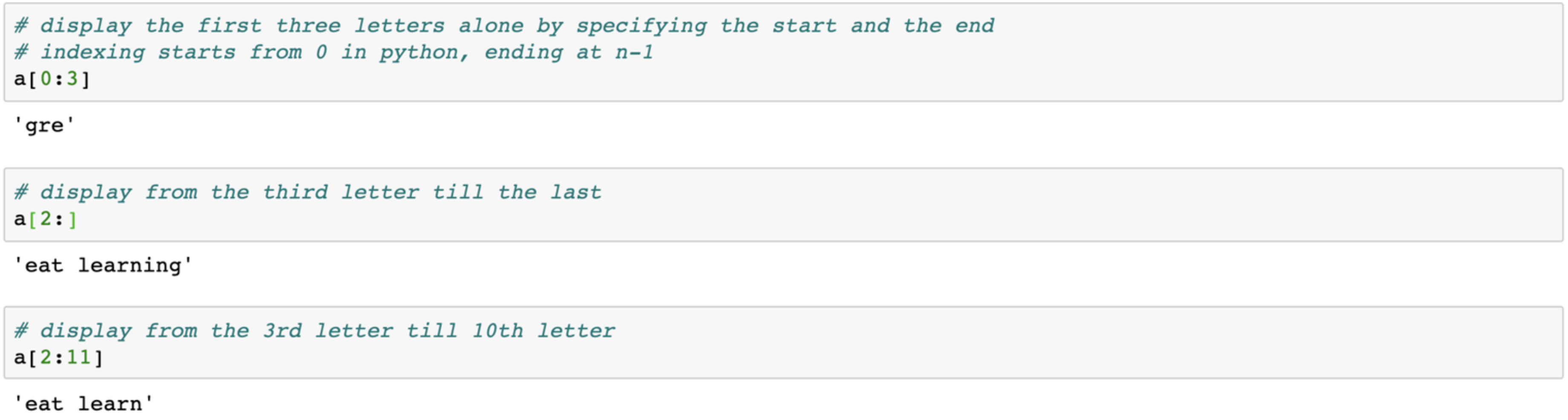
```
'great learning'
```

```
# display all the letter using len()
# len(): gives length of the variable
a[:len(a)]
```

```
'great learning'
```

‘:’ is used to specify range of the sequence to be sliced

Data slicing



```
# display the first three letters alone by specifying the start and the end
# indexing starts from 0 in python, ending at n-1
a[0:3]
```

```
'gre'
```

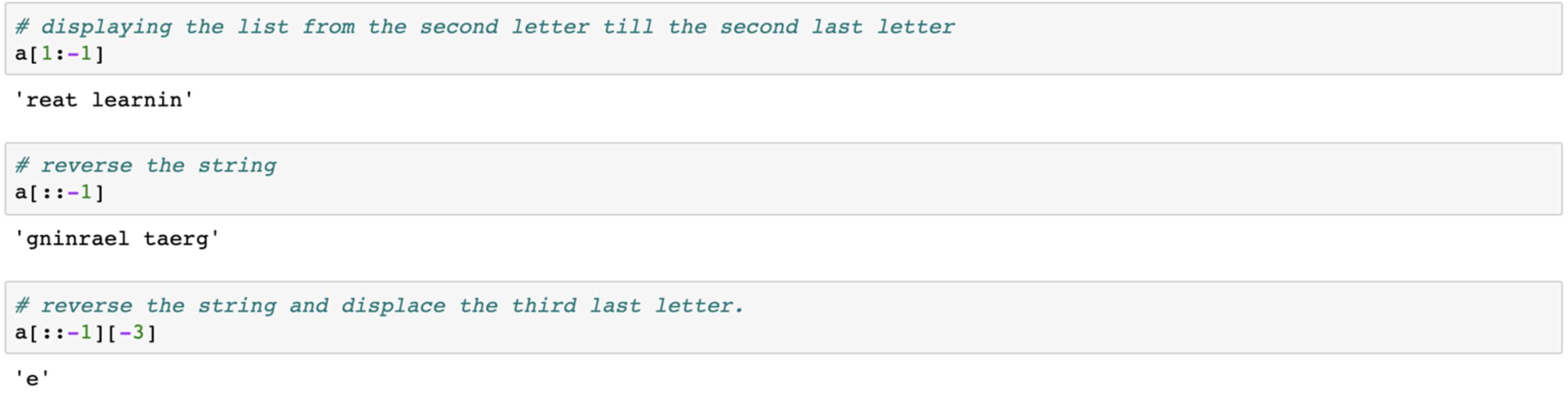
```
# display from the third letter till the last
a[2:]
```

```
'eat learning'
```

```
# display from the 3rd letter till 10th letter
a[2:11]
```

```
'eat learn'
```

Data slicing



```
# displaying the list from the second letter till the second last letter
a[1:-1]
```

```
'reat learnin'
```

```
# reverse the string
a[::-1]
```

```
'gninrael taerg'
```

```
# reverse the string and displace the third last letter.
a[::-1][-3]
```

```
'e'
```

String Handling

```
# replace 'g' with 'G' in the string
a.replace('g','G')
```

```
'Great learninG'
```

```
# convert the string in upper case
a.upper()
```

```
'GREAT LEARNING'
```

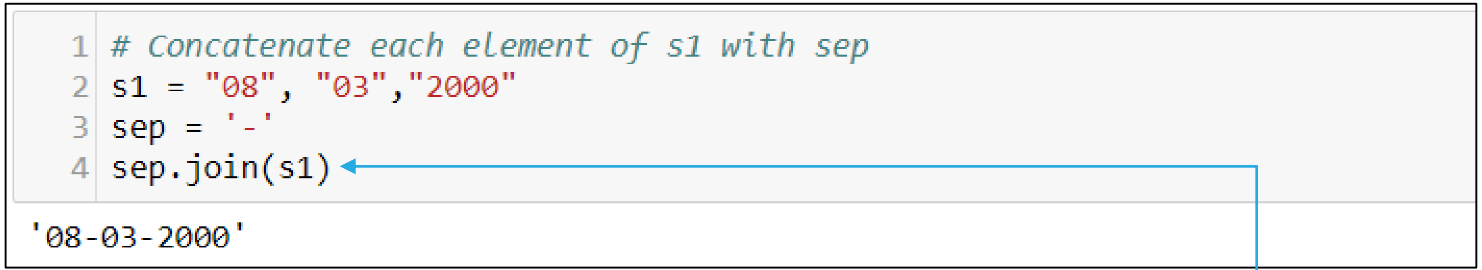
```
# convert the string in lower case
a.lower()
```

```
'great learning'
```

```
# convert the first letter of each string to upper case
a.title()
```

```
'Great Learning'
```

String Handling

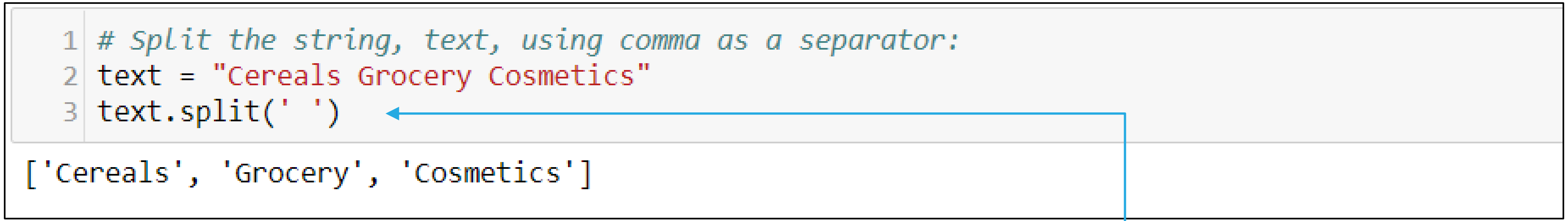


```
# Concatenate each element of s1 with sep
s1 = "08", "03","2000"
sep = '-'
sep.join(s1)
```

```
'08-03-2000'
```

Returns a concatenated string where each character of s1 is separated with sep string which is '-'

String Handling

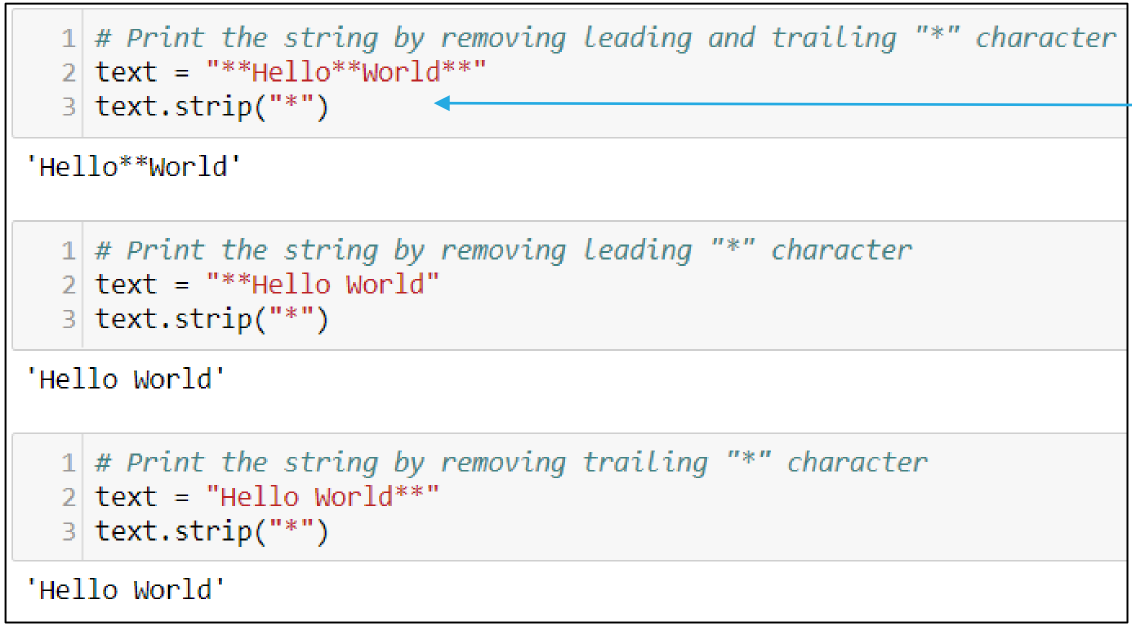


```
# Split the string, text, using comma as a separator:
text = "Cereals Grocery Cosmetics"
text.split(' ')
```

```
['Cereals', 'Grocery', 'Cosmetics']
```

Returns a list containing elements as strings separated by the space character

String Handling



```python
# Print the string by removing leading and trailing "*" character
text = "**Hello**World**"
text.strip("*")
```

```
'Hello**World'
```

```python
# Print the string by removing leading "*" character
text = "**Hello World"
text.strip("*")
```

```
'Hello World'
```

```python
# Print the string by removing trailing "*" character
text = "Hello World**"
text.strip("*")
```

```
'Hello World'
```

Returns a copy of the string with both leading and trailing characters removed (based on the string argument passed)

Python Flow Control

Python Flow Control

Any program has a flow. The flow is the order in which the program’s code executes. The control flow of a Python program is controlled by:

1. Conditional Statements
2. Loops
3. Function Calls

We cover the basics of conditional statements and loops in today’s session

The if-statement

Sometimes we want to execute a code only if a certain condition is true.

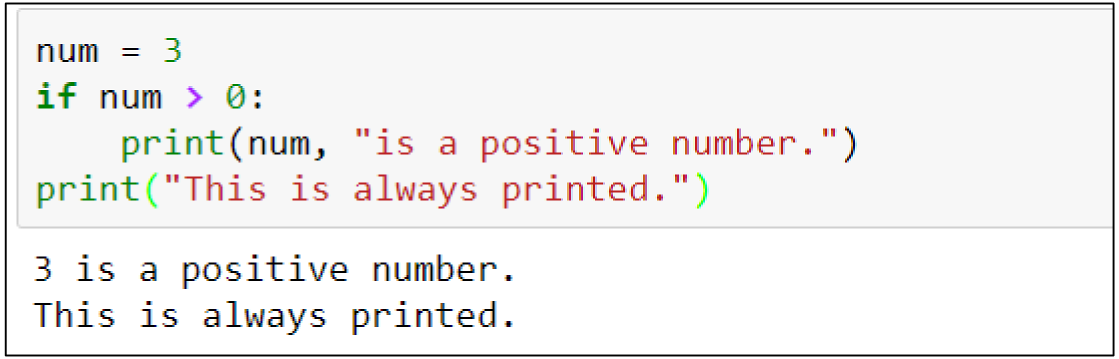
The *if* statement is used in Python for decision making. An "if statement" is written by using the *if* keyword.

Syntax:

if test expression:

statement(s)

The if-statement



```
num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
```

```
3 is a positive number.
This is always printed.
```

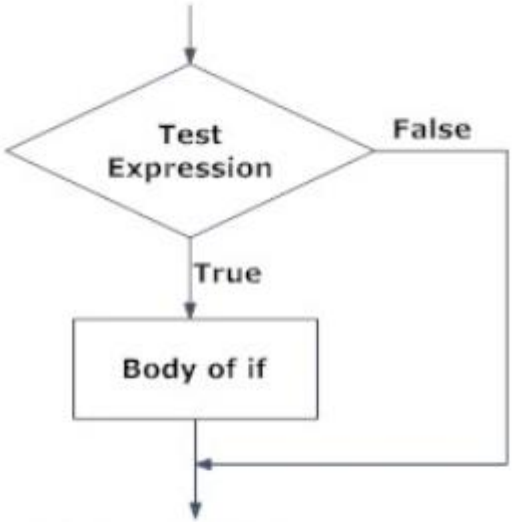


Fig: Operation of if statement

The if-else statement

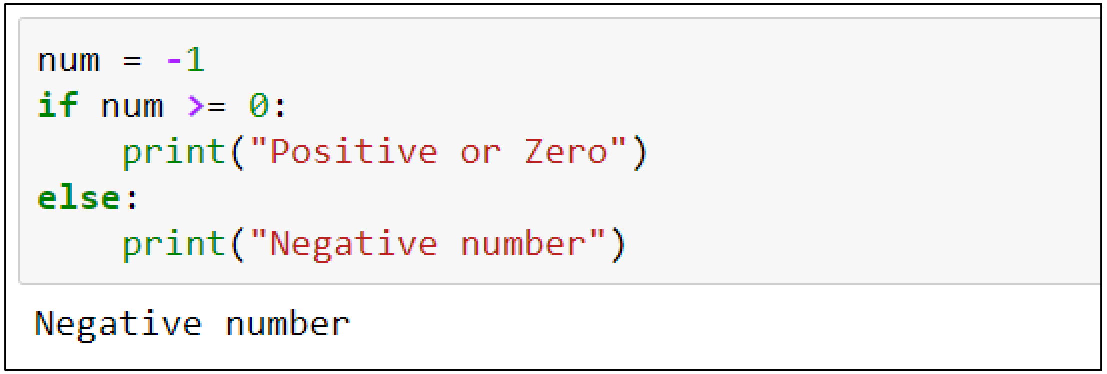
The ‘*if..else*’ statement evaluates a *test expression* and will execute the code that is part of the ‘*if*’ expression if the *test expression True*.

If the test expression is *False*, the code that is part of the *‘else’* expression is executed. Note the indentation that is used to separate the ‘if’ and ‘else’ blocks.

Syntax:

```
if test expression:
    Body of if
else:
    Body of else
```

The if-else statement



```
num = -1
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

```
Negative number
```

Python if..else Flowchart

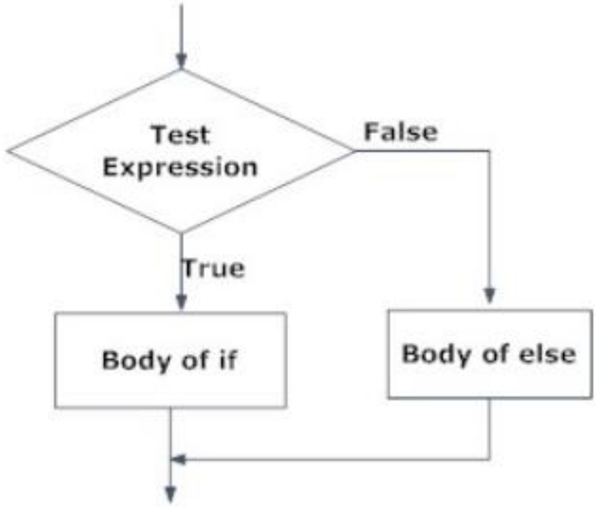


Fig: Operation of if...else statement

The *while* Loop

Loops are used to repeat the execution of a specific block of code.

The ‘*while loop*’ in Python is used to iterate over a block of code as long as the test expression holds true.

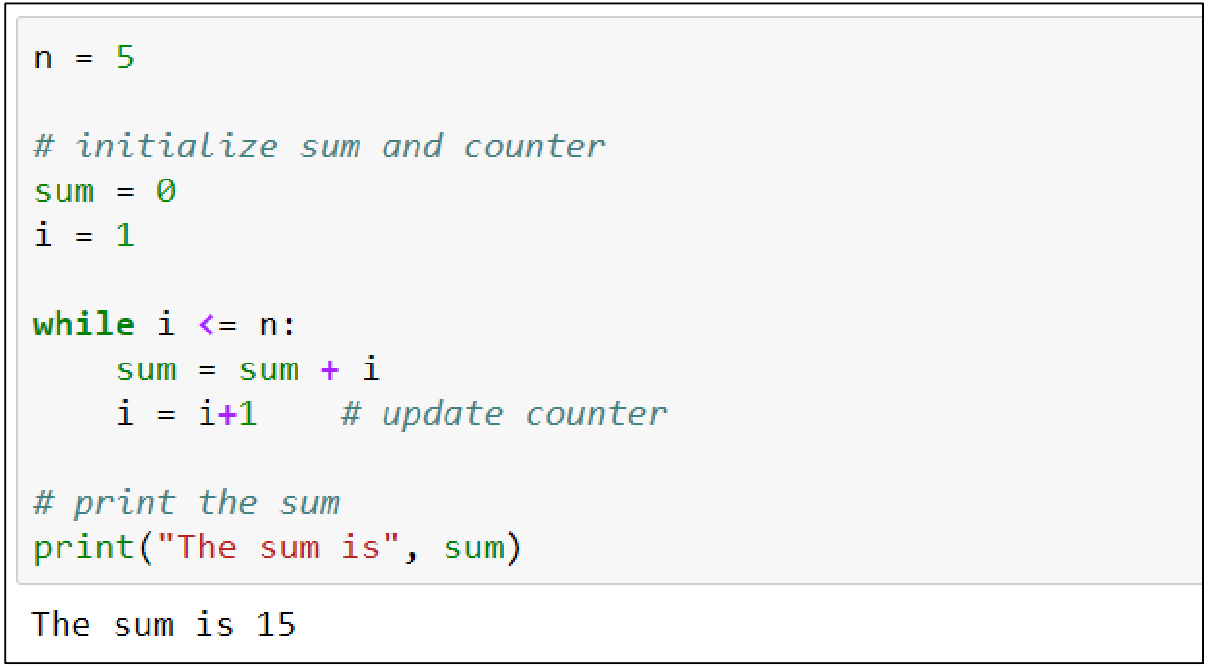
We generally use this loop when the number of times to iterate is not known to us beforehand.

Syntax:

while test_expression:

Body of while

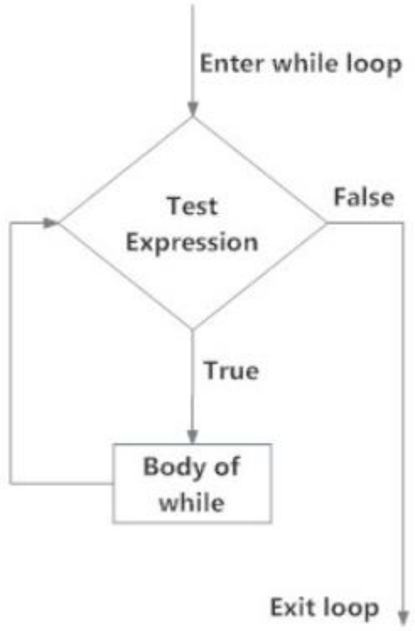
The *while* Loop



```
n = 5

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

```
The sum is 15
```

Flowchart of while Loop



The *for* Loop

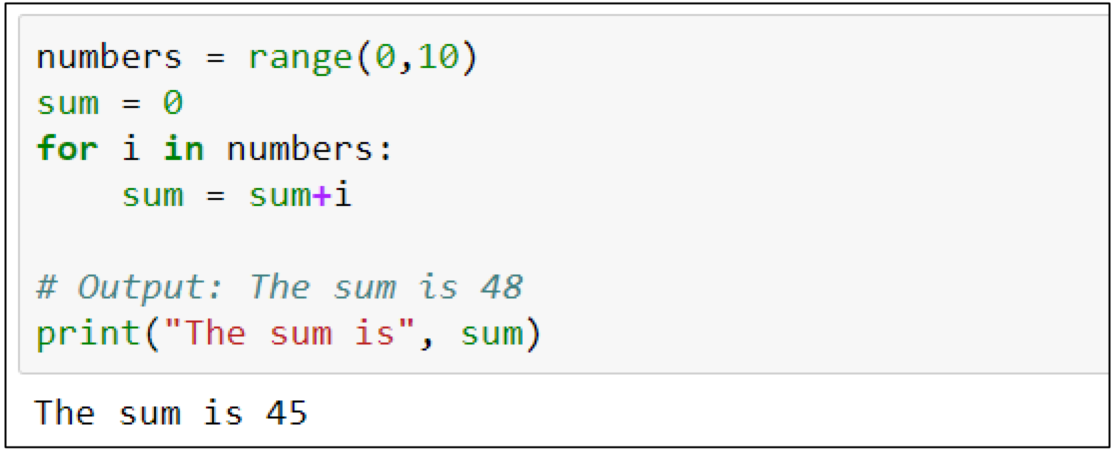
The '*for loop*' in Python is used to iterate over the items of a sequence object like list, tuple, string and other iterable objects.

The iteration continues until we reach the last item in the sequence object. Note the indentation that is used in a '*for loop*' to separate the rest of the code from the 'for loop' syntax

Syntax:

```
for i in sequence:
        Body of for
```

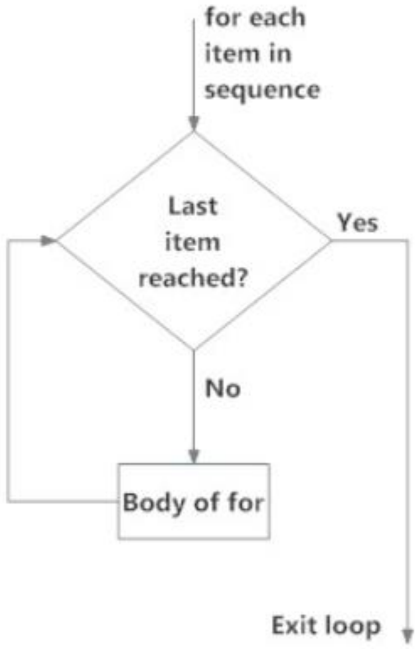
The *for* Loop



```
numbers = range(0,10)
sum = 0
for i in numbers:
    sum = sum+i

# Output: The sum is 48
print("The sum is", sum)
```

```
The sum is 45
```

Flowchart of for Loop



Pseudocode

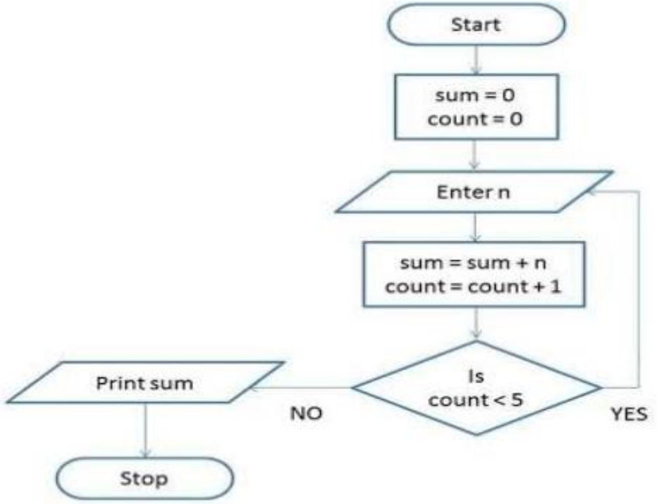
What is Pseudocode?

Pseudocode is a step-by-step written outline of your code that you can gradually transcribe into programming language

Example: Pseudocode

Find the sum of 5 numbers:

1. Set sum=0, count=0
2. Enter the number
3. Add number to sum and store it back to sum
4. Increment count by 1
5. If count < 5, go to step 2 else print total

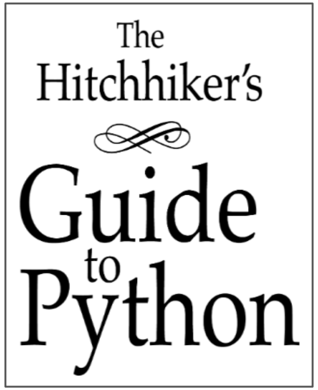


Write pseudocode to calculate sum & average of 10 numbers that you input

There exists a word “Pythonic”? What does it mean?

There are many ways to accomplish the same task in Python, but there is usually one preferred way to do it. This preferred way is called "pythonic."

Read The Hitchhiker’s Guide to Python (https://docs.python-guide.org/writing/style/*).*


The Hitchhiker’s Guide to Python

Thank You