Python

Mrinali Jain

Intro

Coding helps with creativity, logic, and math. It can enable you to do many amazing things; from writing your own websites, to building machines to achieve incredible feats!!! You can even start an entire company with less than 50 dollars in your bank account. In this class, we will write our own games.

Python

Python is the most basic coding language you can learn. It is designed for readability and has a significant use of whitespace where indentations affect your code. Popular websites and apps such as Google, Wikipedia, and Instagram use Python a lot, and others, like Reddit, are made up entirely of Python

Indentation

Python is one of the few languages in which indentation affects your code's outcome.

Ex.

```
if 5 > 2:
    print('five is greater than two!')
```

```
>>> five is greater than two!
```

If you do not indent here, you will get an error message. Indentation shows a block of code. For each block of code, there needs to be a common number of indents in increasing order.

Variables

Variables are words, phrases, or letters to which a value has been assigned. You can also set one variable equal to another

Ex.

```
x = 8          #x is an integer
y = 'John'     #y is a string
Z = x          #z stores the value of x

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

>>> 8
>>> John
>>> 8
```

Variables

Variable names can't start with a number and can only include A - z, 0-9, and the _ character.

To combine both a variable and text, Python uses +.

Ex.

```
myName = 'Rina'
print('My name is' + myName)
```

```
>>> My name is Rina
```

Variables

You can also add the values of variables.

```
1  x = 5
2  y = 7
3  print(x+y)
```

```
12
```

If you try to add a string and a number, you will get an error.

```
1  x = 5
2  y = 'smile'
3  print(x+y)
4
```

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

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`

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview

Strings

Strings are surrounded by either single quotes ‘ hi ’ or double quotes “ hi ”.

For multi-line strings, use 3 single or double quotes. This will preserve your formatting.

```
1  print('''
2  Strings cannot be changed.
3  They are immutable and you
4  can't edit them''')
5
```

```
Strings cannot be changed.
They are immutable and you
can't edit them
```

Strings

You can print part of a string. This is called slicing. You are able to do this because Python can read the character at position i. Indexing starts at 0.

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

llo

The first number is inclusive and the second one is in exclusive. Running this would print the third and the fifth character.

Strings

```
17
 HELLO, WORLD!
 hello, world!
Hello, World!
 Smello, World!
```

```python
b = " Hello, World!   "

# String Length
print(len(b))

# String Methods
print(b.upper())  # returns the string in upper-case
print(b.lower())  # returns the string in lower-case
print(b.strip())  # removes white-space from the beginning and end of the string
print(b.replace("Hell", "Smell"))  # replaces portion of string
                                   with another string
```

Input

You can take input from the user of your code and store it in a variable. To do this, you run the input function.

In the example at the right, *Mrinali* is the input.

```
print('hi')
myName = input('Please enter your name. ')
print()
print('Hello, ' + myName)
```

```
hi
Please enter your name. Mrinali

Hello, Mrinali
```

Booleans

Sometimes, you need to know if an equation is True or False. The returned answer is a boolean. Comparing numbers and variable will return a Boolean.

```
a = 57
b = 31

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

```
b is not greater than a
```

Operators

Operator	Example	Same As
=	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

Assignment Operators

Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Mathematical Operators

Operators (Membership, Identity, and Logical)

Operator	Description	Example
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
in	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 not in y

Operators

Operator	Name	Example
==	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

Comparison Operators

Lists

- 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 and unindexed. No duplicate members.
- Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

Lists

```
food = ["pizza", "burgers", "burritos"]
print(food)
```

```
['pizza', 'burgers', 'burritos']
```

Lists use square brackets.

Lists

You can access items in your set by specifying the index number. The indexing starts at 0.

```
food = ["pizza", "burgers", "burritos"]
print(food[1])
```

```
burgers
```

Pizza has the index value of 0.
Burger has the value 1, and burritos has the value 2. You can also use the range function which is similar to splicing in strings.

Lists

Change Item Values:

```
colors = ["blue", "orange", "red"]
colors[1] = "green"

print(colors)
```

```
['blue', 'green', 'red']
```

Loop Through a List

```
colors = ["blue", "orange", "red"]
for x in colors:
  print(x)
```

```
blue orange red
```

Lists

Check if item is in list.

```
1  ingredients = ['peanut butter', 'jelly', 'bread']
2  if 'peanut butter' in ingredients:
3      print("Peanut Butter is an ingredient.")
```

```
Peanut Butter is an ingredient.
```

Find the length of a list.

```
1  ingredients = ['peanut butter', 'jelly', 'bread']
2  print(len(ingredients))
```

```
3
```

While Loops

While loops allow us to execute a command as long as a condition is true.

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

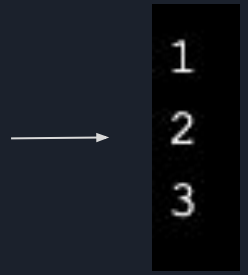
```
1
2
3
4
5
```

As long as i is less than 6 this code will print the value of*i*. Since *i* starts at i, this will happen 5 times.

While Loops

Sometimes you may need to break a loop if a certain case occurs.

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



```
1
2
3
```

Remember that you need to keep your indentations in order!!

While Loops

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



```
1
2
4
5
6
```

After breaking the code, you may want to continue after skipping a certain condition. In this example, we have skipped the 3.

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



```
1
2
3
4
5
i is no longer less than 6
```

If the condition is not fulfilled, you may want something else to happen

For Loops

A for loop is another type of loop that is used to iterate through sequences such as strings and lists which learned about previously. Since we seen its basic uses earlier, we will move on to its more complicated aspects.

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



```
apple
banana
```

For Loops

A very common function that you see used with for loops is the range function.

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



```
2
3
4
5
```

Nested For Loops

A nested for loop is for loop inside a for loop.

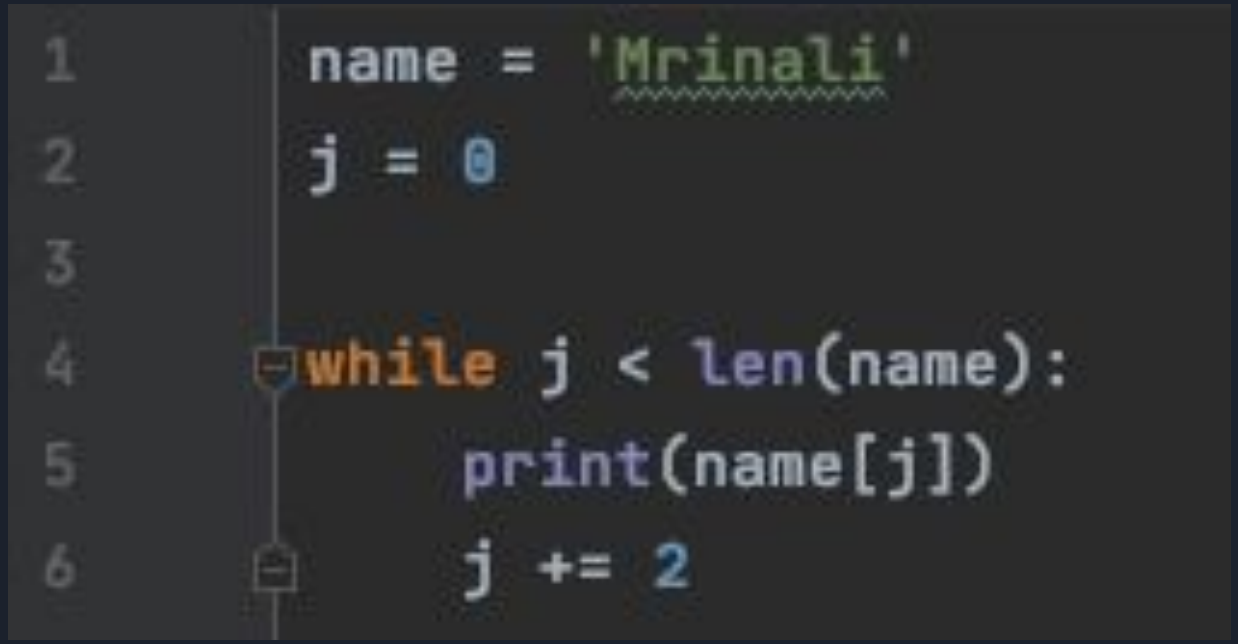
```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

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

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Challenge !!!!!

Print every other letter of your name!

Challenge Solution



Forever Loops

One type of loop is a loop that repeats forever until you break it. For this we will use a while True loop.

Exercise!!!

Create a for loop that that loops through every character of a string.

```
for __ in _____:
    _______________
```

This is the format that you will use. Remember to also define a string.

Function

A function is block of code that runs only when it is called.

Together, we will write last week’s exercise in a function and call it.

We will learn about arguments, which is the input to a function.

```
def myfunc():
    print('hi')

myfunc()
```



```
hi
```

Functions: Returning Values

You can return a value in a function and set it to a variable. This can be used later on in your code. The return command will break the function and store a value.

```
def myfunc(x,y):
    return(x+y)

sum = myfunc(7,9)

print('the sum is ' + str(sum))
```



```
the sum is 16
```

Scope

Certain variables are created within some scopes. For example, a variable defined inside a function is local to the function.

A for loop also defines a variable that is only used on the scope of that loop.

Global variables are defined outside functions and can be used anywhere

Exercise !!!

Create a function that does this. Use a loop to print numbers 1 through any number of my choice. Define any variables of your choice and use any type of loop.

Dictionaries

```
genders = {'mom': 'female', 'dad': 'male','Mrinali': ' female'}
for i in genders:
    print(i)
```

```
mom
dad
Mrinali
```

A dictionary is another type of sequence with keys and values. The values can also be lists.

```
1  genders = {'mom': 'female', 'dad': 'male','Mrinali': 'female'}
2  for i in genders:
3      print(genders[i])
```

```
female
male
female
```