Data Science with Python

Python Fundamentals

Agenda

Introduction to Python

Introduction to Python

Python is an object-oriented, interpreted, high-level programming language. It is general purpose, and we use it to develop GUI and web applications

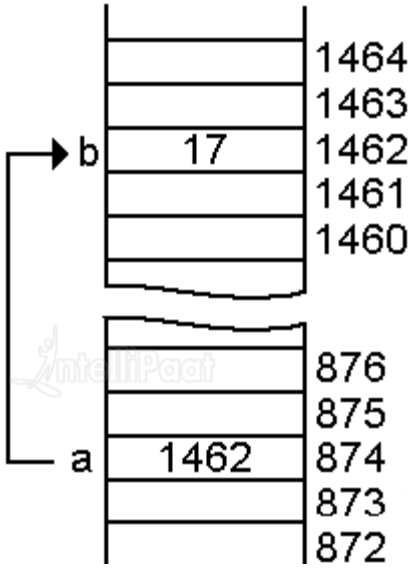
With Python, we can concentrate on the business logic of our code rather than investing a lot of time in common programming tasks

Introduction to Python Variables

Introduction to Python Variables

Whenever we build any application, we need to be able to store some data in our systems memory. We do that using variables.

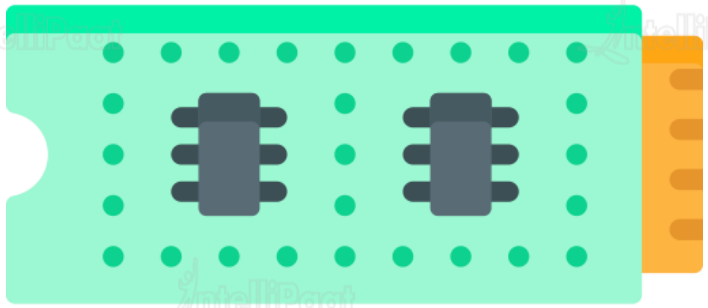
Simply put, variables are used to store and retrieve data from our systems memory



Introduction to Python Variables

We can store value in a variable by either assigning a value to a variable or getting the value as input from user

A variable should start with a letter or an underscore and cannot start with numbers.



Python Variables – Assigning Values

Python Variables – Assigning Values

There are two ways of assigning values to a variable:

1. Assigning a Single Value
2. Multiple Assignment

Python Variables – Assigning Values

Assigning a Single Value

Assigning Multiple Values

Assigning a single value to a variable:

```
a = 10
name = 'Victor'
salary = 2000.23
print (a)
print (name)
print (salary)
```

```
10
Victor
2000.23
```

Python Variables – Assigning Values

Assigning a Single Value

Assigning Multiple Values

Assigning multiple values:

```
a=b=c=10
x,y,z=10,20,30
print(y)
print(z)
print(a)
```

```
20
30
10
```

Python Variables – Getting User Input

Python Variables – Getting User Input

To get input from user in python and then assign that value to a python variable we need to use the input function

To use the input function we need to also show a prompt to user asking them to enter a value

Variable Assignment

Prompt

```
In [ ]: # Get user's name as input
        name = input("Enter you name: ")
```

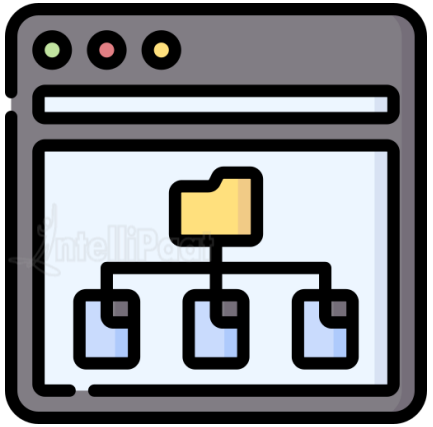
Hands On – Getting User Input

Introduction to Data Types

Introduction to Data Types

A Data Type is simply a piece of information associated with a variable to indicate to the interpreter what type of data is stored in a variable, e.g.: Number, Text etc.

This information can then be used to determine what kind of operations are valid on a variable or multiple variables



Introduction to Data Types

In Python there are six types of Data Types

Integer

Float

String

Boolean

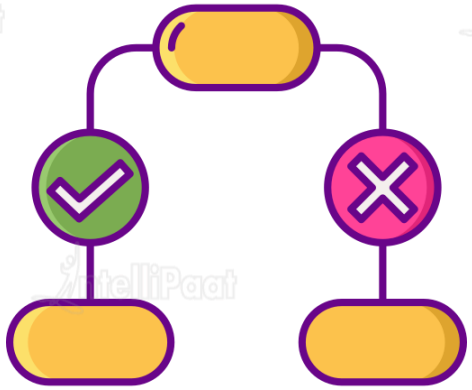
Hands-on: Data Types

Conditional Statements

Conditional Statements

Sometimes in an application we have to perform certain tasks if a given condition is true e.g. Load profile if user is logged in etc.

To accomplish this in code we use conditional statements



Conditional Statements

Conditional Statements are used to change the flow of execution when a particular condition evaluates to True or False

There are three kinds of conditional statements

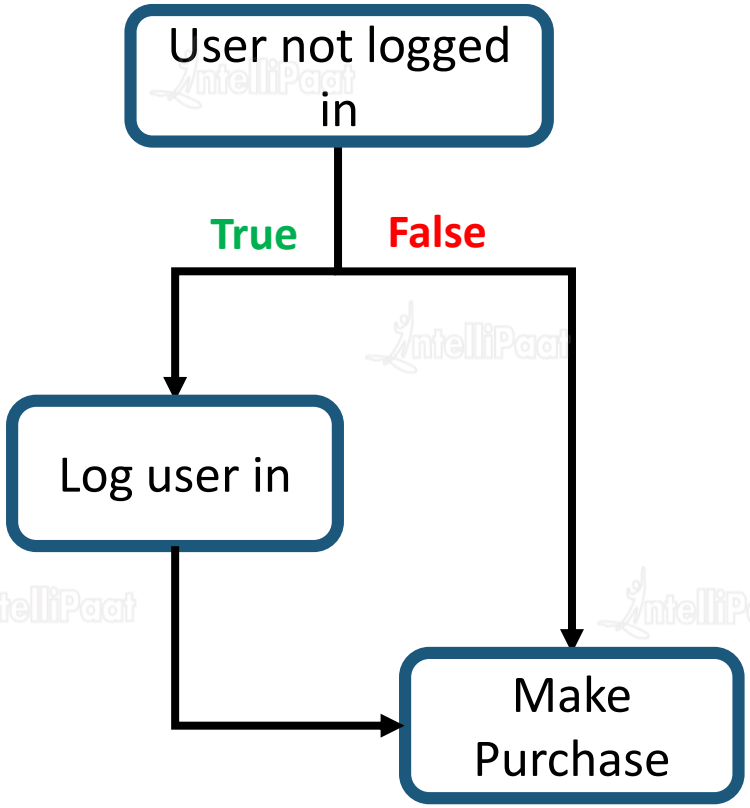
If

If - else

If - elif

Conditional Statements - If

An if statement is used execute some code if certain condition evaluates to be True

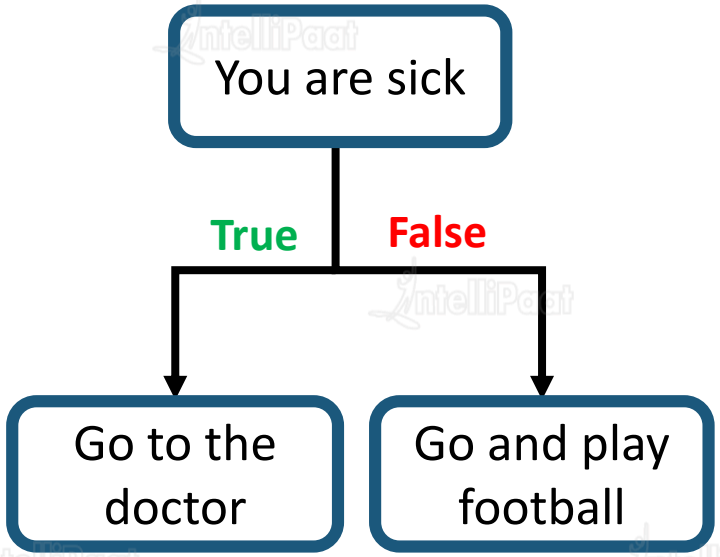



```python
isUserLoggedIn = False

if not isUserLoggedIn:
    print("Redirect to login")

print("Make Purchase")
```

```
Redirect to login
Make Purchase
```

Conditional Statements - If - else

An if else statement is used execute some code if certain condition evaluates to be True and some other code if statement evaluates to false

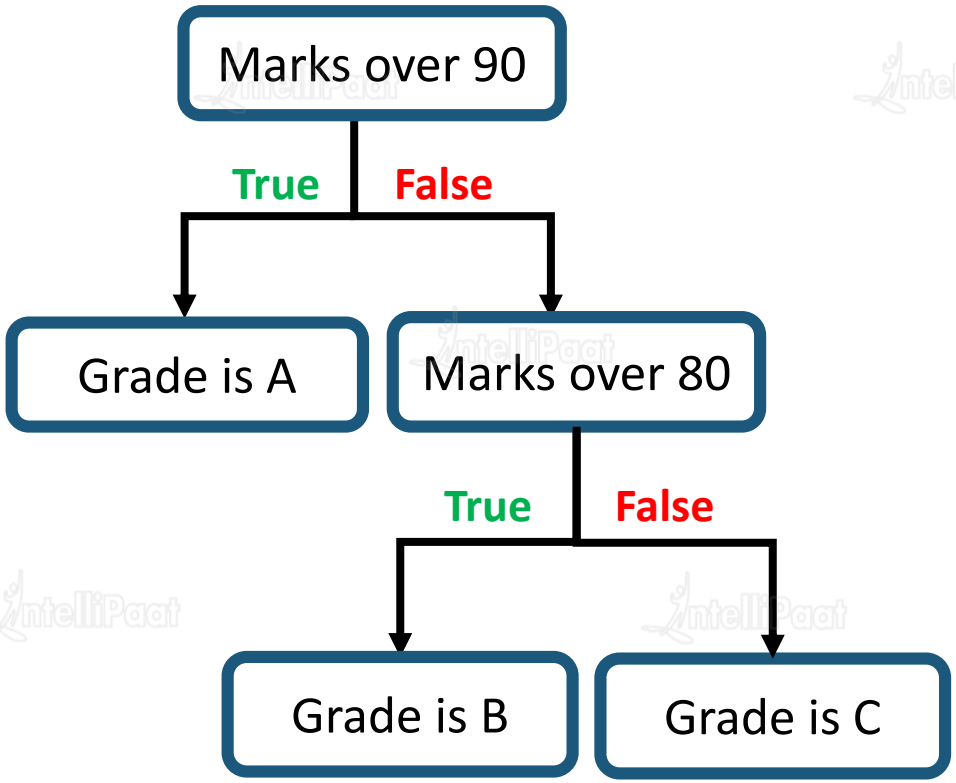



```python
youAreSick = True

if youAreSick:
    print("Go To The Doctor")
else:
    print("Go and play football")
```

```
Go To The Doctor
```

Conditional Statements - If - elif

The elif keyword used when we have multiple conditions and want to check them one by one




```python
marks = 95

if marks >= 90:
    print("Grade is A")
elif marks >= 80:
    print("Grade is B")
else:
    print("Grade is C")
```

```
Grade is A
```

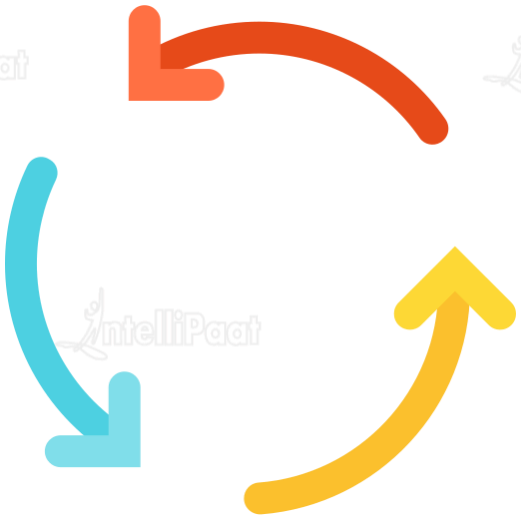
Hands-on: Checking if a number is odd

Looping Statements

Looping Statements

When programming there are times when you have to perform certain tasks for a number of times, e.g. printing a name 100 times

You can copy and paste some code multiple time to do that or you can instruct you application to do it a number of times using loops



Looping Statements

Looping is the process in which we have a some code that gets executed repeatedly until a particular condition is satisfied

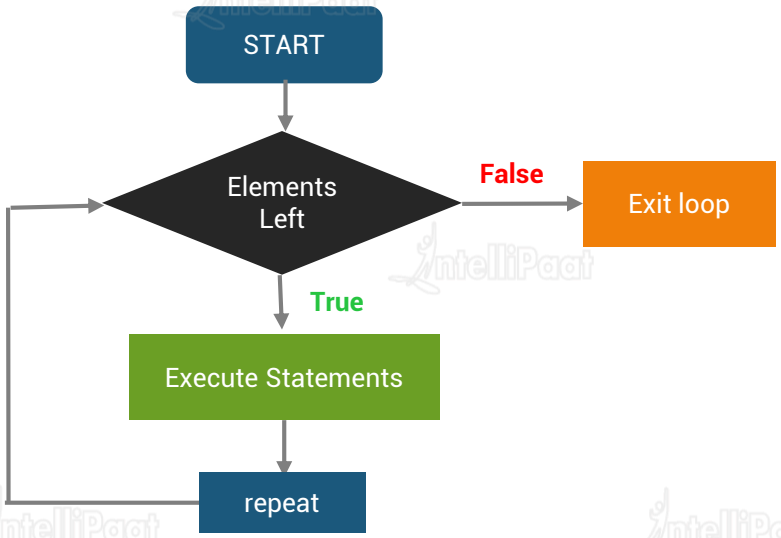
There are two kinds of loops used in python

For

While

Looping Statements - For Loop

The for statement is used to loop over a group or collection of data

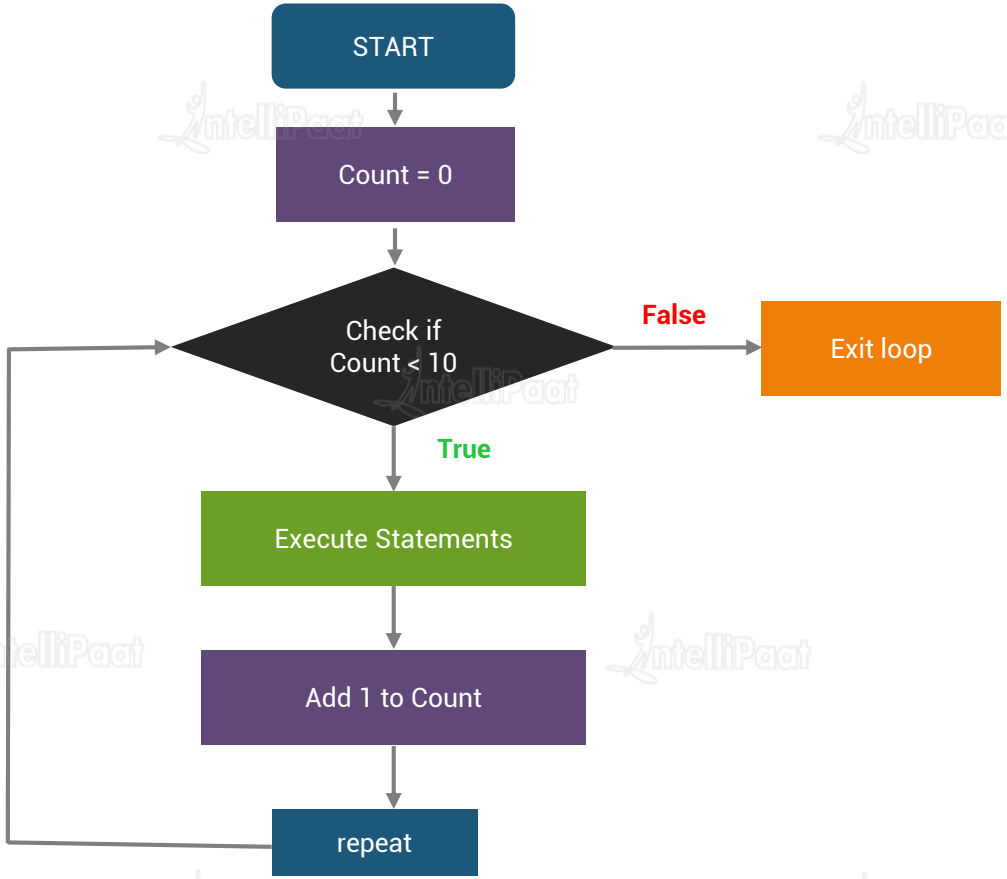



```python
for num in range(0, 10):
    print(num)
```

```
0
1
2
3
4
5
6
7
8
9
```

Looping Statements - While Loop

The while statement simply loops until a condition is evaluates to False




```python
count = 0
while count < 10:
    print(count)
    count += 1
```

```
0
1
2
3
4
5
6
7
8
9
```

Hands-on: Printing the multiplication table of a given number

List

Lists

In Python Lists are used to store collection of in a sequential order e.g. Items in a wish list of a customer

Since they are stored in a sequential order, a special number based on their position in the list called index is assigned to them and they are accessed using these index values

Data	J	O	H	N	N	Y
Index	0	1	2	3	4	5

Lists

These indexes start from zero and are assigned in an increasing order i.e. from zero to n – 1 where n is number of items in that lists

These lists can store data of multiple data types, e.g. Integer, String, Float etc.

Data	J	O	H	N	N	Y
Index	0	1	2	3	4	5

List Operations

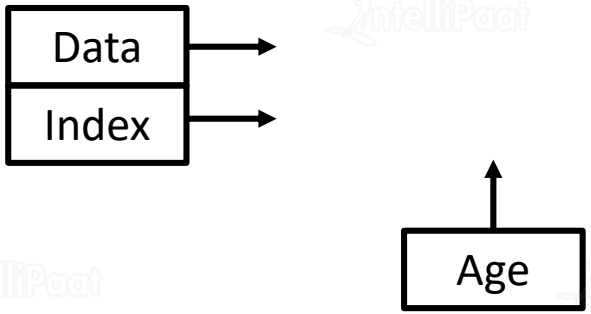
List Operations - Create

Create | Initialize | Append | Delete

To create an empty list you can use the list function or use empty brackets

```
names = list()
age = []
print(age)
```

```
[]
```



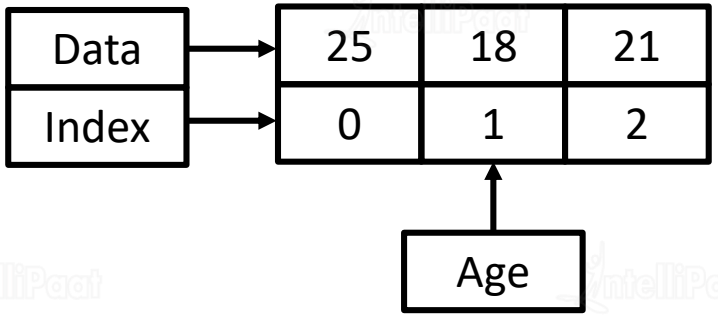
List Operations - Initialize

Create | **Initialize** | Append | Delete

You can initialize a list with some data while creating the list, by passing values inside the list function or brackets

```
age = [25, 18, 21]
print(age)
```

```
[25, 18, 21]
```



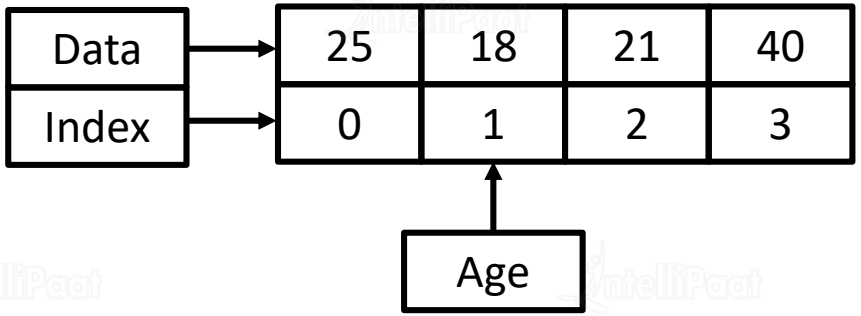
List Operations - Append

Create | Initialize | **Append** | Delete

You can add data to the end of list by calling the append method on the lists

```
age.append(40)
print(age)
```

```
[25, 18, 21, 40]
```



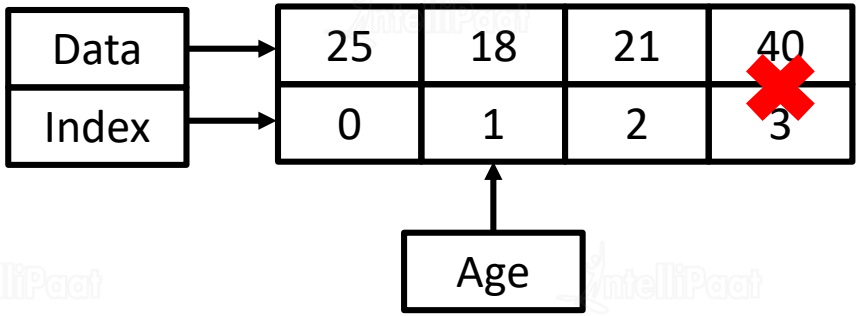
List Operations - Delete

Create | Initialize | Append | Delete

You can delete an element in a list by calling the pop method on it. Pop will delete the element at the end of the list

```
age.pop()
print(age)
```

```
[25, 18, 21]
```



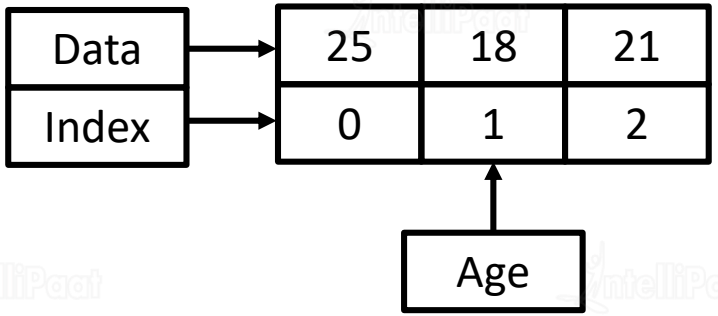
List Operations - Delete

Create | Initialize | Append | Delete

You can delete an element in a list by calling the pop method on it. Pop will delete the element at the end of the list

```
age.pop()
print(age)
```

```
[25, 18, 21]
```



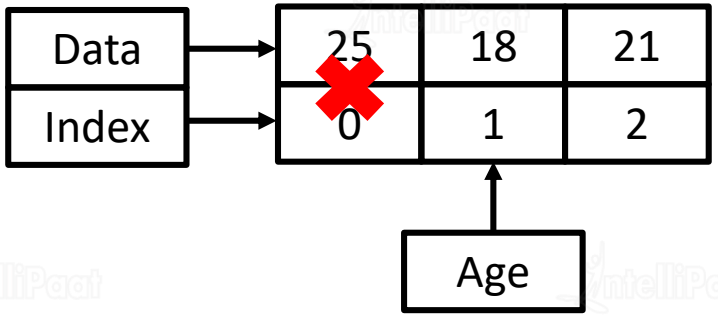
List Operations - Delete

Create | Initialize | Append | Delete

You can also delete an element at a particular index, by passing the index in as parameters in the function call

```
age.pop(0)
print(age)
```

```
[18, 21]
```



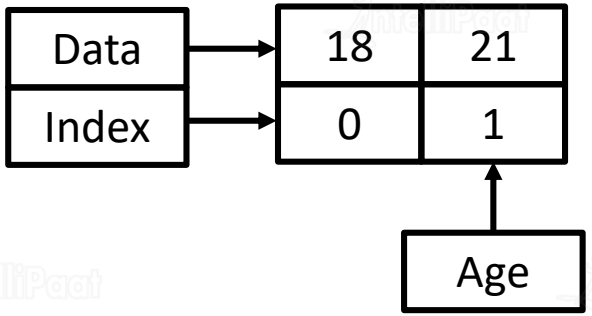
List Operations - Delete

Create | Initialize | Append | Delete

Notice that this ends up changing the index of all the elements that occur after the deleted index

```
age.pop(0)
print(age)
```

```
[18, 21]
```



Tuples

Tuples

In Python just like Lists, Tuples are also used to store collection of in a sequential order, the difference is that it is immutable

Immutable means that once you create a tuple, you cant make changes to it, i.e. add items, remove items, swap items etc.

Data	J	O	H	N	N	Y
Index	0	1	2	3	4	5

Tuples

Tuples are especially useful when you have a collections of data which you do not wish to change in your application e.g. Days in a Week

Tuples will throw an error if you try and change them, so you wont be able to accidently change the tuple

Data	J	O	H	N	N	Y
Index	0	1	2	3	4	5

Tuples

Like lists, elements in tuples are also accessed using their indexes

Tuples can also store data of multiple types such as Integer, Float Boolean etc.

Data	J	O	H	N	N	Y
Index	0	1	2	3	4	5

Tuple Operations

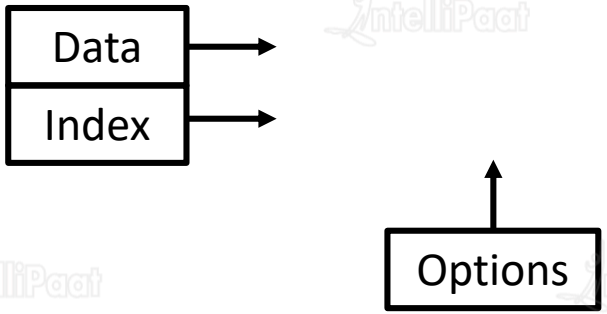
Tuple Operations - Create

Create | Initialize | Search | Slice

To create an empty tuple you can use the tuple function or use empty parenthesis

```
options = ()
names = tuple()
print(options)
```

```
()
```



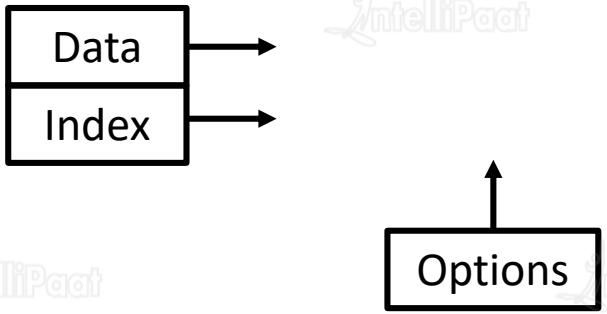
Tuple Operations - Create

Create | Initialize | Search | Slice

To initialize a tuple with some values you can either use the tuple function or use the comma syntax

```
options = ()
names = tuple()
print(options)
```

```
()
```



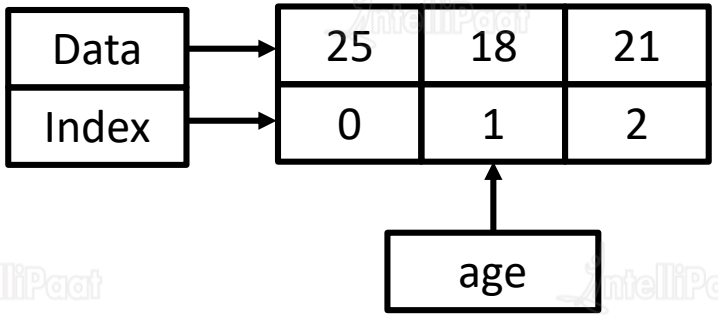
Tuple Operations - Initialize

Create | **Initialize** | Search | Slice

In the tuple function you need to pass in a other data structure like list and it will return a new tuple with all the values from that data structure

```
age_list = [25, 18, 21]
age = tuple(age_list)
print(age)
```

```
(25, 18, 21)
```



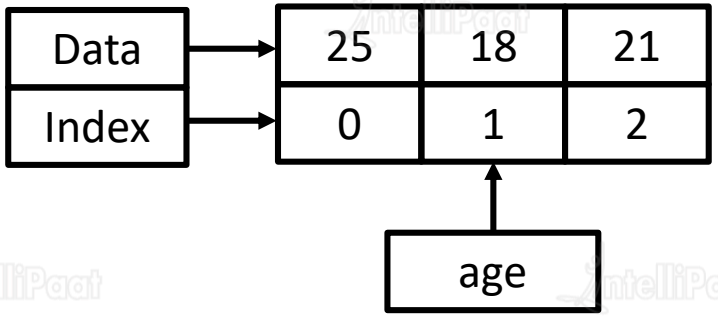
Tuple Operations - Initialize

Create | **Initialize** | Search | Slice

To use the comma syntax you need to have a few values separated by commas and assign them to a variable

```
age = 25, 18, 21
print(age)
```

```
(25, 18, 21)
```



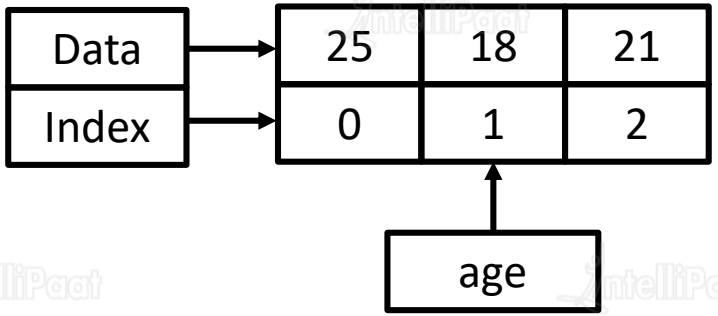
Tuple Operations - Search

Create | Initialize | **Search** | Slice

You can check either if a particular element exists in the tuple or you can check at what index does a particular element exist

```
print(age)
```

```
(25, 18, 21)
```



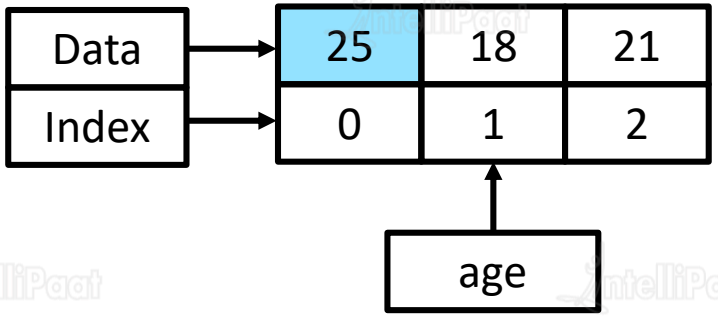
Tuple Operations - Search

Create | Initialize | **Search** | Slice

To check if a particular element exists in the tuple we use the membership operator (using the 'in' keyword), which returns True if element exists and False if it does not

```
21 in age
```

```
True
```



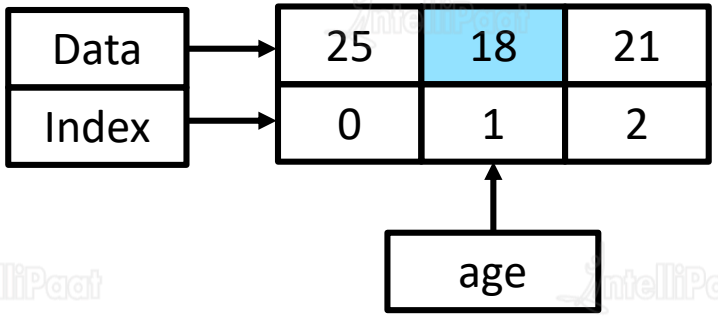
Tuple Operations - Search

Create | Initialize | Search | Slice

To check if a particular element exists in the tuple we use the membership operator (using the 'in' keyword), which returns True if element exists and False if it does not

```
21 in age
```

```
True
```



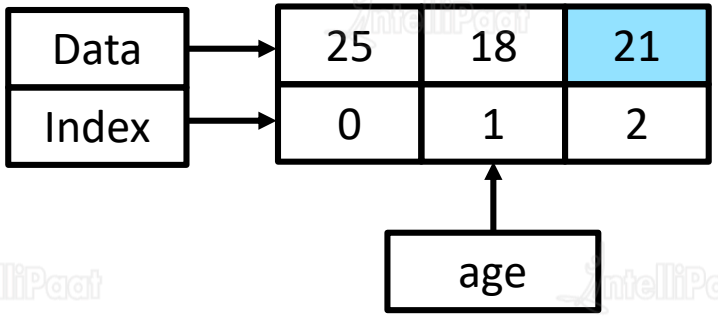
Tuple Operations - Search

Create | Initialize | **Search** | Slice

To check if a particular element exists in the tuple we use the membership operator (using the 'in' keyword), which returns True if element exists and False if it does not

```
21 in age
```

```
True
```



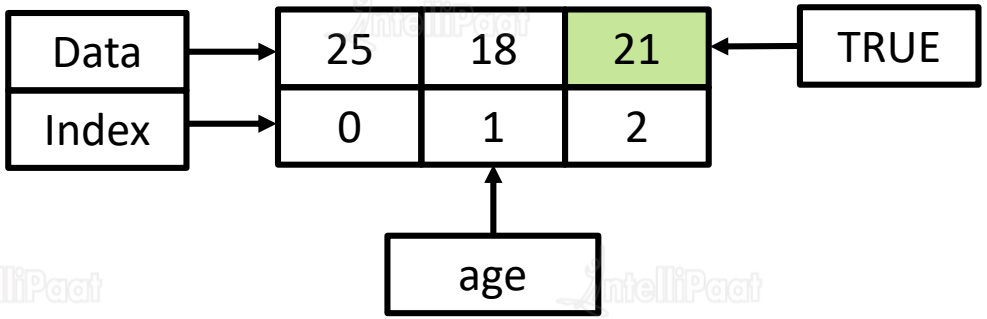
Tuple Operations - Search

Create | Initialize | **Search** | Slice

To check if a particular element exists in the tuple we use the membership operator (using the 'in' keyword), which returns True if element exists and False if it does not

```
21 in age
```

```
True
```



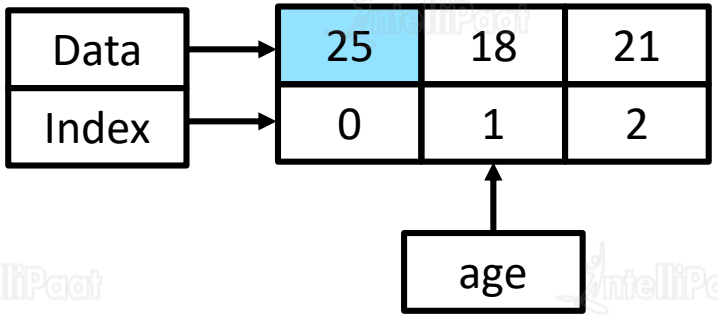
Tuple Operations - Search

Create | Initialize | Search | Slice

You can also use the index method to find the index of a particular element in a tuple. It returns the index if the element is found if not then throws an error

```
age.index(21)
```

```
2
```



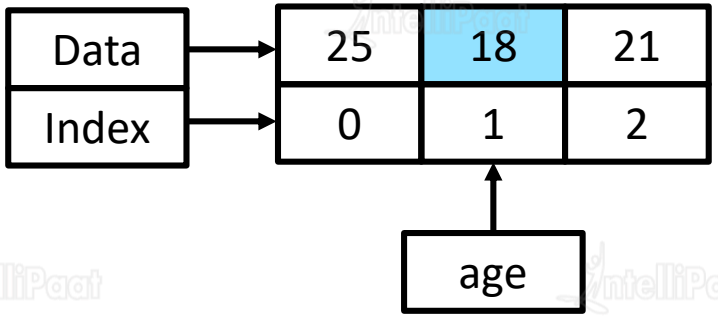
Tuple Operations - Search

Create | Initialize | Search | Slice

You can also use the index method to find the index of a particular element in a tuple. It returns the index if the element is found if not then throws an error

```
age.index(21)
```

```
2
```



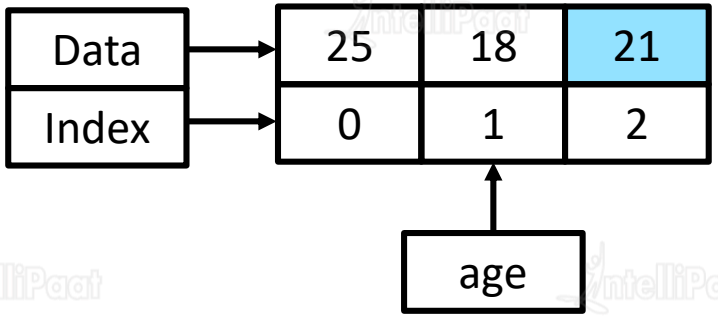
Tuple Operations - Search

Create | Initialize | Search | Slice

You can also use the index method to find the index of a particular element in a tuple. It returns the index if the element is found if not then throws an error

```
age.index(21)
```

```
2
```



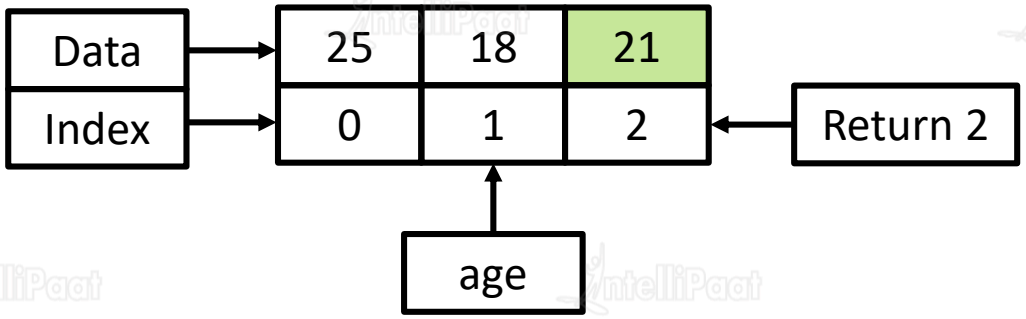
Tuple Operations - Search

Create | Initialize | **Search** | Slice

You can also use the index method to find the index of a particular element in a tuple. It returns the index if the element is found if not then throws an error

```
age.index(21)
```

```
2
```



Tuple Operations - Slice

Create | Initialize | Search | Slice

Slicing is used to get a contiguous portion of a list. For example, if wish to get a copy of elements from index 1 to 3

```
age = 25, 18, 21, 40, 50, 45
print(age)
```

```
(25, 18, 21, 40, 50, 45)
```

Data	25	18	21	40	50	45
Index	0	1	2	3	4	5

Tuple Operations - Slice

Create | Initialize | Search | Slice

To slice a list we need to provide the index from where you wish to start the slice and index before which the slice ends like age[1:4]

```
print(age[1:4])
```

```
[18, 21, 40]
```

Data	25	18	21	40	50	45
Index	0	1	2	3	4	5

Tuple Operations - Slice

Create | Initialize | Search | **Slice**

Do note that we use 4 instead of 3 to indicate that slice needs to stop before index 4

```
print(age[1:4])
```

```
[18, 21, 40]
```

Data	25	18	21	40	50	45
Index	0	1	2	3	4	5

Dictionaries

Dictionaries

Dictionaries like sets are unordered collections of data but they store key value pairs, i.e. two associated values

Much like sets dictionaries are also great for checking membership of keys when you have as key value pairs

Key	Value
A	1
B	2
C	3

Dictionary Operations

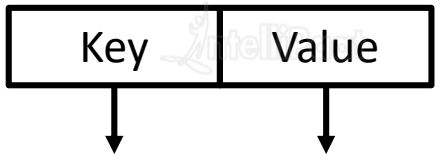
Dictionary Operations - Create

Create | Initialize | Add | Remove | Search

To create an empty dictionary you can either use the dict function or use the curly braces syntax

```
names = {}
age = dict()
print(age)
```

```
{}
```



Dictionary Operations - Initialize

Create | **Initialize** | Add | Remove | Search

To initialize a dictionary with some values you can either pass some nested data structures to dict function or use the curly braces syntax

```
names = {}
age = dict()
print(age)
```

```
{}
```

Key	Value

Dictionary Operations - Initialize

Create | **Initialize** | Add | Remove | Search

You can create a dictionary with data structures like lists or tuples. These data structures need to contain either lists or tuples with 2 values each

```
age_list = [['a', 1], ['b', 2], ['c', 3]]
age = dict(age_list)
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Key	Value
A	1
B	2
C	3

Dictionary Operations - Initialize

Create | **Initialize** | Add | Remove | Search

Notice that each value is either a list or tuple of size 2 in which index zero is the key index one is the value

```
age_list = [['a', 1], ['b', 2], ['c', 3]]
age = dict(age_list)
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Key	Value
A	1
B	2
C	3

Dictionary Operations - Initialize

Create | **Initialize** | Add | Remove | Search

You can also use curly braces syntax by separating key value pairs using commas and keys values using colon e.g. **{key1 : value1, key2: value2}**

```
age = {'a' : 1, 'b' : 2, 'c' : 3}
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Key	Value
A	1
B	2
C	3

Dictionary Operations - Add

Create | Initialize | **Add** | Remove | Search

To add a new key value pair by using this syntax:
dict_name[key] = value

```
age['d'] = 5
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 5}
```

Key	Value
A	1
B	2
C	3
D	5

Dictionary Operations - Add

Create | Initialize | **Add** | Remove | Search

Do note that if add values using an existing key it will overwrite the previous value

```
age['d'] = 4
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Key	Value
A	1
B	2
C	3
D	4

Dictionary Operations - Add

Create | Initialize | Add | **Remove** | Search

To remove an element just use the del keyword with the key dictionary name e.g. del age[‘d’]

```
age['d'] = 4
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Key	Value
A	1
B	2
C	3
D	4

Dictionary Operations - Add

Create | Initialize | Add | **Remove** | Search

To remove an element just use the del keyword with the key dictionary name e.g. del age[‘d’]

```
del age['d']
print(age)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Key	Value
A	1
B	2
C	3
D	4

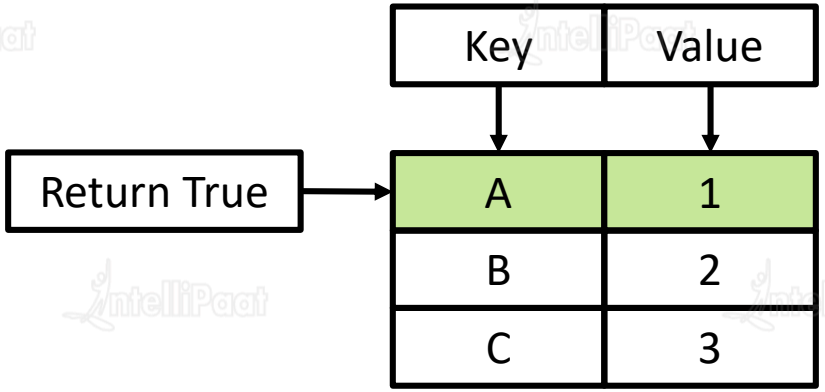
Dictionary Operations - Remove

Create | Initialize | Add | Remove | Search

To search a value you need to use the membership operator with the key,
If the key is found it returns true else it returns false

```
'a' in age
```

```
True
```



Creating a Function

Creating a Function

To create a function we first need to understand the syntax of a function

A function definition can have multiple parts to it

Name

Parameters

Return statement

Name

Parameters

```
def add_two_numbers(x, y):
    print("Adding two numbers")
    return x + y
```

Return

Creating a Function - Name

When a function is being defined we give it a name so that it can be referenced later

The name of the function is like a naming a variable and follows all the same rules

Name

Parameters

Return statement

Name

Parameters

```
def add_two_numbers(x, y):
    print("Adding two numbers")
    return x + y
```

Return

Creating a Function - Parameters

Parameters or Function Arguments are the variables or data that we want our function to work on, e.g. numbers to be added

You can have any number of parameters be accepted in your function or no parameters if your function does not need it

Name

Parameters

Return statement

```
def add_two_numbers(x, y):
    print("Adding two numbers")
    return x + y
```

Name

Parameters

Return

Creating a Function - Parameters

A return statement is used to return the answer computer by your function

You can have no return value at the end by simply omitting the return statement in case you so not wish your function to return a value

Name

Parameters

Return statement

Name

Parameters

```
def add_two_numbers(x, y):
    print("Adding two numbers")
    return x + y
```

Return

Types of Functions

Types of Functions

There are two types of functions

User Defined Function

Built In Defined Function

Types of Functions – User Defined Functions

User-defined Functions

Built-in Functions

These functions are defined and used by the developers who are writing the code and wish to solve a problem for which there isn't a function in the standard library

```
def find_max(nums):
    result = nums[0]
    for x in nums:
        if x > result:
            result = x
    return result
```

```
find_max([1, 8, 2])
```

```
8
```

Functions in Python

User-defined Functions

Built-in Functions

These functions come built into the language as part of the standard library for example, functions like print, input etc.

abs(): Returns the absolute value of a number

all(): Returns True if all items in an iterable object are true

any(): Returns True if any item in an iterable object is true

ascii(): Returns a readable version of an object and replaces non-ASCII characters with an 'escape' character

bin(): Returns the binary version of a number

bool(): Returns the Boolean value of a specified object

Hands-on: Functions

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor