BEINEX

Full Stack Internship Python

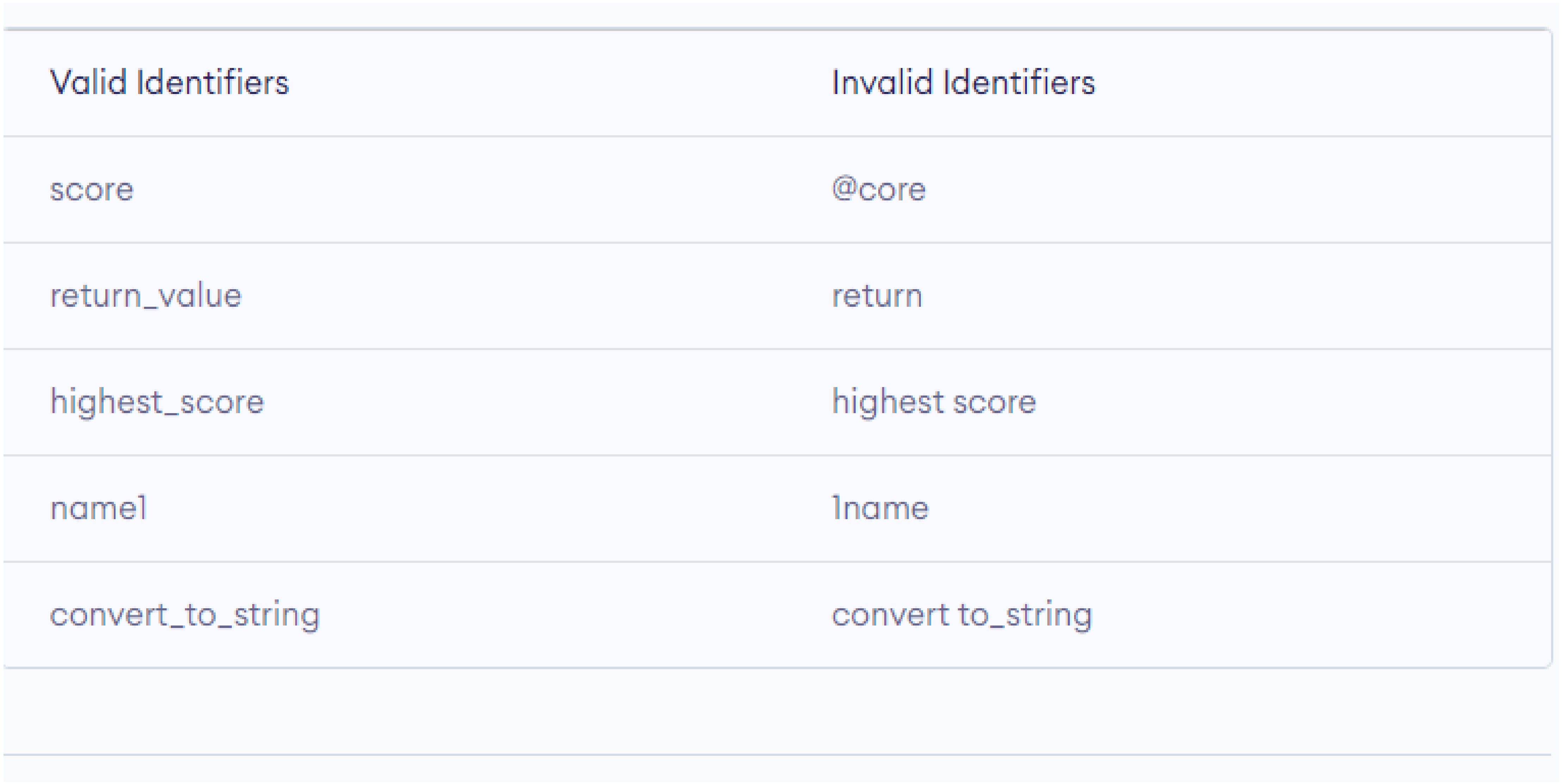
Identifiers in Python

- **Identifier** is a user-defined name given to a variable, function, class, module, etc. , The identifier is a combination of character digits and an underscore.
- They are case-sensitive i.e., ‘num’ and ‘Num’ and ‘NUM’ are three different identifiers in python.
- It is a good programming practice to give meaningful names to identifiers to make the code understandable.
- We can also use the Python string isidentifier() method to check whether a string is a valid identifier or not.

Rules for Naming Python Identifiers

- It cannot be a reserved python keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore (_).
- It should not contain any special character other than an underscore (_).

Examples of Python Identifiers



Valid Identifiers	Invalid Identifiers
score	@core
return_value	return
highest_score	highest score
name1	1name
convert_to_string	convert to_string

Python String isidentifier() Method

- **Python String isidentifier() method** is used to check whether a string is a valid identifier or not.
- The method returns True if the string is a valid identifier, else returns False.

Example :

```
string = "Coding_101"
print(string.isidentifier())


Output : True
```

```
string = "54Geeks0for0Geeks"
print(string.isidentifier())


Output : False
```

Python Variables

- Variables are containers for storing data values.
- A variable is created the moment you first assign a value to it.

Eg:

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

- Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Eg:

```
x = 4       # x is of type int
x = “John" # x is now of type str
print(x)
```

Casting

- To specify the data type of a variable, this can be done with casting.

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

Get the type

- Get the type of a variable with the type() function

Eg :
```
x=5              print(type(x))
y = “John”       print(type(y))
```

Single or Double Quotes

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

Eg :

```
x = "John"
# is the same as
x = 'John’
```

Case-Sensitive

- Variable names are case-sensitive.

Eg :

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

Multi Words Variable Names

Variable names with more than one word can be difficult to read.
There are several techniques you can use to make them more readable:

Camel Case

- Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

- Each word starts with a capital letter:

```
MyVariableName = "John“
```

Snake Case

- Each word is separated by an underscore character:

```
my_variable_name = "John"
```

Assign Multiple Values

Many Values to Multiple Variables

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

Eg :

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

One Value to Multiple Variables

- assign the *same* value to multiple variables in one line:

Eg:

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

Unpack a Collection

- Collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Eg :

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

Output Variables

- The Python print() function is often used to output variables.

Eg :

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

- In the print() function you output multiple variables, separated by a comma:

Eg:

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

Global Variables

- Variables that are created outside of a function are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

Eg : *Create a variable outside of a function, and use it inside the function*

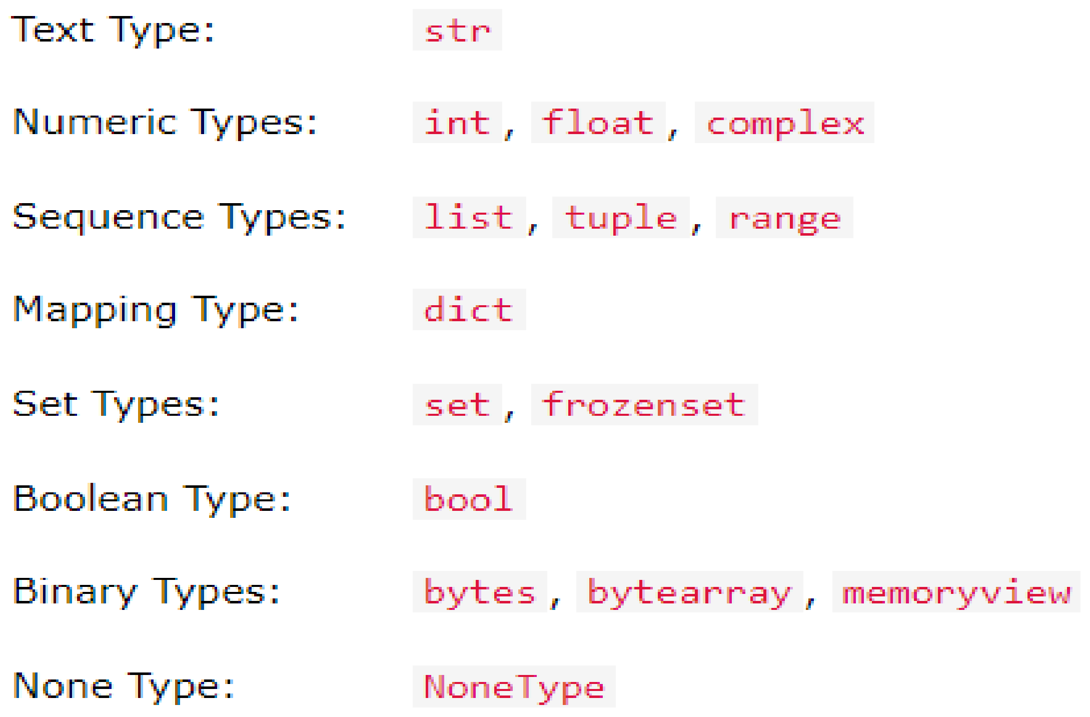
```
x = "awesome"

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

myfunc()
```

Output: *Python is awesome*

Data Types

Built-in Data Types

- In programming, data type is an important concept.
- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:



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`

Example	**Data Type**
`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`	NoneType