Python: Session 1

The Interactive Shell:

What's this interactive shell thing you're running?

It's like this: Python leads a double life. It's an interpreter for scripts that you can run from the command line or run like applications, by double−clicking the scripts. But it's also an interactive shell that can evaluate arbitrary statements and expressions. This is extremely useful for debugging, quick hacking, and testing.

Example:

```
>>> 1 + 1

2

>>> print ('hello world')

hello world

>>> x = 1

>>> y = 2

>>> x + y

3
```

The Python interactive shell can evaluate arbitrary Python expressions, including any basic arithmetic expression. The interactive shell can execute arbitrary Python statements, including the print statement. You can also assign values to variables, and the values will be remembered as long as the shell is open.

Compiler Vs Interpreter:

Both compiler and interpreters do the same job which is converting higher level programming language to machine code. However, a compiler will convert the code into machine code (create an .exe) before program run. Interpreters convert code into machine code when the program is run. Compiler will parse or analyses all of the language statements for its correctness at once. If incorrect, throws an error. But in interpreter Source statements executed line by line DURING Execution.

Statically typed language:

A language in which types are fixed at compile time. Most statically typed languages enforce this by requiring you to declare all variables with their datatypes before using them. Java and C are statically typed languages.

Dynamically typed language:

A language in which types are discovered at execution time; the opposite of statically typed. VBScript and Python are dynamically typed, because they figure out what type a variable is when you first assign it a value.

Strongly typed language:

A language in which types are always enforced. Java and Python are strongly typed. If you have an integer, you can't treat it like a string without explicitly converting it.

Weakly typed language:

A language in which types may be ignored; the opposite of strongly typed. VBScript is weakly typed. In VBScript, you can concatenate the string '12' and the integer 3 to get the string '123', then treat that as the integer 123, all without any explicit conversion.

So Python is both dynamically typed (because it doesn't use explicit datatype declarations) and strongly typed (because once a variable has a datatype, it actually matters).

Data types:

1. **Dictionary:**
   One of Python's built−in datatypes is the dictionary, which defines one−to−one relationships between keys and values.
   Example: Defining a Dictionary

```
>>> d = {"server":"mpilgrim", "database":"master"}
>>> d
{'server': 'mpilgrim', 'database': 'master'}
>>> d["server"]
 'mpilgrim'
>>> d["mpilgrim"]
 KeyError: mpilgrim
```

   Modifying dictionary:

```
>>> d["database"] = "pubs"
>>> d
{'server': 'mpilgrim', 'database': 'pubs'}
>>> d["uid"] = "sa"
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'pubs'}
```

   Deleting

```
>>> del d[‘uid’]
>>> d
{'server': 'mpilgrim', 'database': 'pubs'}
```

2. **Tuple:**

A tuple is an immutable list. A tuple can not be changed in any way once it is created.

Example. Defining a tuple

```
>>> t = ("a", "b", "mpilgrim", "z", "example")
```

```
>>> t

('a', 'b', 'mpilgrim', 'z', 'example')

>>> t[0]

'a'

 >>> t[−1]

'example'

>>> t[1:3]

('b', 'mpilgrim')
```

3. **Sets:**

   A set is a collection which is unordered and unindexed.

   ```
   >>> thisset = {"apple", "banana", "cherry"}
   >>> print(thisset)
   ```

   ‘apple’, ‘banana’, ‘cherry’

   You cannot access items in a set by referring to an index, since sets are unordered the items has no index.But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

   ```
   >>>thisset = {"apple", "banana", "cherry"}

   >>>for x in thisset:
   >>>       print(x)
   ```

   ‘apple’

   ‘banana’

   ‘cherry’

4. **Lists**

   Example:Defining a List

   ```
   >>> li = ["a", "b", "mpilgrim", "z", "example"]
   >>> li
   ['a', 'b', 'mpilgrim', 'z', 'example']
   >>> li[0]
   'a'
   ```

   Example. Adding Elements to a List

   ```
   >>> li ['a', 'b', 'mpilgrim', 'z', 'example']
   >>> li.append("new")
   >>> li
   ```

```
['a', 'b', 'mpilgrim', 'z', 'example', 'new']
>>> li.insert(2, "new")
>>> li
['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new']
>>> li.extend(["two", "elements"])
>>> li
['a', 'b', 'new', 'mpilgrim', 'z', 'example', 'new', 'two', 'elements']
```

Difference between extend and append

```
>>> li = ['a', 'b', 'c']
>>> li.extend(['d', 'e', 'f'])
>>> li
['a', 'b', 'c', 'd', 'e', 'f']
>>> li = ['a', 'b', 'c']
>>> li.append(['d', 'e', 'f'])
>>> li
['a', 'b', 'c', ['d', 'e', 'f']]
```

Example Negative List Indices

```
>>> li ['a', 'b', 'mpilgrim', 'z', 'example']
>>> li[−1]
'example'
>>> li[−3]
'mpilgrim'
```

Example
Slicing a List

```
>>> li
['a', 'b', 'mpilgrim', 'z', 'example']
>>> li[1:3]
['b', 'mpilgrim']
>>> li[1:−1]
['b', 'mpilgrim', 'z']
>>> li[0:3]
['a', 'b', 'mpilgrim']

>>> li
['a', 'b', 'mpilgrim', 'z', 'example']
>>> li[:3]
['a', 'b', 'mpilgrim']
>>> li[3:]
['z', 'example']
>>> li[:]
['a', 'b', 'mpilgrim', 'z', 'example']
```