3.DataType

April 17, 2020

Data Types

raw data --> process it

Data Type

why we do classification ?

house

- kitchen
- store room
- study room
- dinning room

Data Process

store

delete

add

remove

search

update

sort

Personal Data Type

- D Drive
  - data movies, text, music, images, books

```
    movie --> action with comedy


    D Drive                                 action
                                            action comedy
        documents                           comedy
                              hindi -->     action
                                            romance
                              hollywood
        movies   --->
                              south
        music

        video

        images

        books
Boys rooms


Girls rooms
```

text , numbers

```
linear

        arrays, link-list, stack, queue

non-linear

       tree, graph
```

0.0.1 C language

```
basic data --> int, float, char

data structure  --> array, link-list, tree, graph

    store

    access

    search

    update

    delete
```

sort

0.0.2 Python built-in Advance Data Type

1. Numbers

- Strings
- List
- Tuple
- Dict
- Set
- FrozenSet
- Boolean
- None

lexicals, tokens

words --> vocabulary (keywords)

grammer --> syntax --> arrangement of words

[2]:
```
import keyword
```

[16]:
```
print(*keyword.kwlist, sep=', ')
print("\ntotal keywords are : ", len(keyword.kwlist))
```

False, None, True, and, as, assert, async, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

total keywords are : 35

[6]:
```
import builtins
```

[15]:
```
c = 0
for func in dir(builtins):
    if not func[0].isupper() and not '_' in func:
        print(func, end=', ')
        c += 1
print("\n\nTotal Functions in Python are: ", c)
```

abs, all, any, ascii, bin, bool, breakpoint, bytearray, bytes, callable, chr, classmethod, compile, complex, copyright, credits, delattr, dict, dir, display, divmod, enumerate, eval, exec, filter, float, format, frozenset, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip,

```
Total Functions in Python are:  72
```

[18]:
```
print("Hello World")
```

```
Hello World
```

[19]:
```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

don't talk show me your code

[ ]:
```

```

```
language  --> tokens, lexicons
```

[ ]:
```

```

[21]:
```
name = 'sachin' # statement

# name --> use define keyword, identifier, container, variable
# = ---> operator --> is used store right side value into left side container
# 'sachin' --> data --> string
```

[22]:
```
'sachin' = name # errors
```

```
  File "<ipython-input-22-181160e394bf>", line 1
```

```
    'sachin' = name
             ^
SyntaxError: can't assign to literal
```

[27]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

[23]:
```
print(hello world)
```

```
  File "<ipython-input-23-6135c6da936c>", line 1
    print(hello world)
                ^
SyntaxError: invalid syntax
```

[28]:
```
print(hello, world)
```

```
    ␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-28-137915367199> in <module>
----> 1 print(hello, world)


NameError: name 'world' is not defined
```

[25]:
```
print("hello") # "hello" --> data --> string
```

```
hello
```

[24]:
```
print(hello) # hello --> container, variable
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-24-1cd80308eb4c> in <module>
----> 1 print(hello)


NameError: name 'hello' is not defined
```

[26]:
```
hello = "Hello World"
print(hello)
```

```
Hello World

80 % debugging rest is coding
```

[ ]:
```

```

[ ]:
```

```

1. Numbers

- Strings
- List
- Tuple
- Dict
- Setm
- FrozenSet
- Boolean
- None

```
1. Numbers

   Integers

   Float

   Complex
```

[29]:
```
x = 5 # 5 is int so is x

y = 6.02 # 6.02 is category of float values that's why y will float

z = 56 + 4j # j is complex value so is z
```

[30]:
```
print(type(x), x)
print(type(y), y)
print(type(z), z)
```

```
<class 'int'> 5
<class 'float'> 6.02
<class 'complex'> (56+4j)
```

[31]:
```
5 + 6
```

[31]:
```
11
```

[32]:
```
5 / 6
```

[32]:
```
0.8333333333333334
```

[33]:
```
x = 125ab
```

```
  File "<ipython-input-33-a0c5cd57107a>", line 1
    x = 125ab
            ^
SyntaxError: invalid syntax
```

[34]:
```
x = "hello world"
y = 'this is called string'
```

[35]:
```
print(type(x))
```

```
<class 'str'>
```

[36]:
```
print(type(y))
```

```
<class 'str'>
```

[37]:
```
print(x)
```

```
hello world
```

[38]:
```
print(y)
```

```
this is called string
```

[39]:
```
s  = "He said, "she is beautiful.""
print(s)
```

```
  File "<ipython-input-39-fa3302162a66>", line 1
    s  = "He said, "she is beautiful.""
                    ^
SyntaxError: invalid syntax
```

[40]:
```
s  = 'He said, "she is beautiful."'
print(s)
```

```
He said, "she is beautiful."
```

[41]:
```
s = 'that's your problem, you do this all time'
print(s)
```

```
  File "<ipython-input-41-ce64637b6735>", line 1
    s = 'that's your problem, you do this all time'
              ^
SyntaxError: invalid syntax
```

[42]:
```
s = "that's your problem, you do this all time"
print(s)
```

```
that's your problem, you do this all time
```

[43]:
```
s = "that's it, he said, "Now you will see results.""
print(s)
```

```
  File "<ipython-input-43-c8a452a012de>", line 1
    s = "that's it, he said, "Now you will see results.""
                             ^
SyntaxError: invalid syntax
```

[46]:
```
s = """that's it, he said, "Now you will see results."""
print(s)
```

```
  File "<ipython-input-46-145f3f064dac>", line 1
    s = """that's it, he said, "Now you will see results."""
                                                           ^
SyntaxError: EOL while scanning string literal
```

[47]:
```
s = """that's it, he said, "Now you will see results."""
print(s)
```

```
that's it, he said, "Now you will see results.
```

\ --> escape character / special character

\n --> new line
\t --> tab (4 white or 8 white editor and language)

[48]:
```
print("Hello World")
```

```
Hello World
```

[49]:
```
print("Hello\n\n\t\t\tWorld")
```

```
Hello


                        World
```

[ ]:
```

```

[52]:
```
s = "that's it, he said, \"Now you will see results.\""
print(s)
```

```
that's it, he said, "Now you will see results."
```

[54]:
```
s = 'that\'s it.'
print(s)
```

```
that's it.
```

how will you print \

[55]:
```
print("\")
```

```
  File "<ipython-input-55-dff1232de400>", line 1
    print("\")
          ^
SyntaxError: EOL while scanning string literal
```

[60]:
```
"\""
```

[60]:
```
'"'
```

[61]:
```
print("\tHello\t\tworld") # \t --> tab
```

```
	Hello		world
```

[62]:
```
print('that\'s it')
```

```
  File "<ipython-input-62-48343e48c5b2>", line 1
    print('that's it')
                 ^
SyntaxError: invalid syntax
```

[63]:
```
print('\')
```

```
  File "<ipython-input-63-eaac87876c3b>", line 1
    print('\')
               ^
SyntaxError: EOL while scanning string literal
```

[56]:
```
print("\\")
```

```
\
```

[57]:
```
s = """
Hello World,
    How are you
That's Awesome
"""
```

[58]:
```
print(s)
```

```
Hello World,
    How are you
That's Awesome
```

[64]:
```
"""
hello world
this is
a multi
line comment"""

# single line comments

print("Hello")
```

```
Hello
```

```
Type of Data Type

    sequential orderd data type

    unordered data type

    collection type

    iterable type

    map type

    mutable

    immutable
```

hello world how are **you** that's good

```
**hello world**
how are **you**
    that's good
```

```
collection , sequential, iterable ---> more than one
```

Data Type

```
(Primitive) Primary Type --> atomic --> numbers, char, int, float

(Non-Primitive) Secondary Type --> colleciton of primary types, list, dict, tuple, class

strings --> non-primitive
```

s = "Hello World"

```
H
e
l
l
o
```

```
W
o
r
l
d
```

strings are collection of characters

[65]:
```
print("Hello World")
```

```
Hello World
```

[66]:
```
print(*"Hello World") # * iteration
```

```
H e l l o   W o r l d
```

[67]:
```
print("H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d")
```

```
H e l l o   W o r l d
```

iterable, collection, sequence

[68]:
```
print(*1234) # numbers are primitive (atomic)
```

```
 ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-68-6fde31235bb6> in <module>
   ----> 1 print(*1234)


       TypeError: print() argument after * must be an iterable, not int
```

[69]:
```
lang = [ 'java', 'c', 'c++', 'ruby', 'perl', 'python']
```

[70]:
```
print(lang)
```

```
['java', 'c', 'c++', 'ruby', 'perl', 'python']
```

[71]:
```
print(*lang)
```

```
java c c++ ruby perl python
```

[73]:
```
for char in "Python":
    print(char, end=' ')
```

```
P y t h o n
```

[74]:
```
print(*"Python")
```

```
P y t h o n
```

[75]:
```
lang
```

[75]:
```
['java', 'c', 'c++', 'ruby', 'perl', 'python']
```

[77]:
```
for item in lang:
    print(item, end=" ") # iteration
```

```
java c c++ ruby perl python
```

[78]:
```
print(*lang)
```

```
java c c++ ruby perl python
```

except numbers everything is iterable means we can access one by one element

[79]:
```
s = "Hello World"
```

iterable, collection, sequence

Ordered Data Type --> strings, list, tuple

Unordered Data Type --> dictionary, set, frozenset

[81]:
```
s = "Hello World"
```

”H” ”e” ”l” ”l” ”o” ” ” ”W” ”o” ”r” ”l” ”d”

0 1 2 3 4 5 6 7 8 9 10

index –> 0

[85]:
```
for i,c in enumerate(s):
    print(f"{i:>2}", repr(c))
```

```
 0 'H'
 1 'e'
 2 'l'
 3 'l'
 4 'o'
 5 ' '
 6 'W'
```

```
 7 'o'
 8 'r'
 9 'l'
10 'd'
```

array –> char array –> list char

[86]:
```
lang
```

[86]:
```
['java', 'c', 'c++', 'ruby', 'perl', 'python']
```

[87]:
```
for i,c in enumerate(lang):
    print(f"{i:>2}", repr(c)) # index, loc
```

```
 0 'java'
 1 'c'
 2 'c++'
 3 'ruby'
 4 'perl'
 5 'python'
```

if data type is ordered you can use indexing and slicing

[88]:
```
s = "Hello World"
```

[89]:
```
s[0] # [index] --> indexing --> to acces any single char or item from a␣
↪sequence or collection
```

[89]:
```
'H'
```

[90]:
```
s[1]
```

[90]:
```
'e'
```

[91]:
```
s[6]
```

[91]:
```
'W'
```

[92]:
```
len(s)
```

[92]:
```
11
```

[94]:
```
s[-3]
```

[94]:
```
'r'
```

[93]:
```
print("Hello world")
```

```
Hello world
```

to access sub string we use slicing

syntax

str[start:end:step]

[95]:
```
s = "Hello World"
```

[96]:
```
s[0:5:1]
```

[96]: 'Hello'

[97]:
```
#s[start:end:step]
s[6:11:1]
# world
```

[97]: 'World'

[98]:
```
#s[start:end:step]
#HloWrd
s[0:11:2]
```

[98]: 'HloWrd'

[ ]:
```
s = "Hello World"
```

```
start --> optional
end --> optional
step --> compulsory  (default = +1)

                step --> direction jump
                       + --> +ive direction  (Left to Right)
                       - --> -ive direction  (Right to Left)
```

[99]:
```
s
```

[99]: 'Hello World'

[101]:
```
s[1:6]
```

[101]: 'ello '

[102]:
```
s[1:6:2]
```

[102]: 'el '

[103]:
```
s[:]
```

[103]: 'Hello World'

[104]:
```
s = "Hello World"
```

[106]:
```
s[:5]
```

[106]:
```
'Hello'
```

syntax

s[start:end:step]

[108]:
```
s[:5:-1]
```

[108]:
```
'dlroW'
```

[109]:
```
s = "Hello World"
```

[111]:
```
s[:]
```

[111]:
```
'Hello World'
```

```
start = 0
end = 11
step = +1
      + --> Left to Right
```

[112]:
```
s[6:]
```

[112]:
```
'World'
```

[113]:
```
s[6::-1]
```

[113]:
```
'W olleH'
```

[ ]:
```
s = "Hello World"
```

[114]:
```
s[::-1]
```

[114]:
```
'dlroW olleH'
```

[115]:
```
s[::-2]
```

[115]:
```
'drWolH'
```

[116]:
```
s[::2]
```

[116]:
```
'HloWrd'
```

[117]:
```
s = "Hello World"
```

[118]:
```
s[0:5:7]
```

[118]:
```
'H'
```

[ ]:
```

```