Syntax

June 27, 2022

1 Python Syntax

1.0.1 Line Structure

[2]:
```
a = input("Enter a number :")
print(a)
```

```
Enter a number :10
10
```

[5]:
```
print("Hello Everyone")
```

```
Hello Everyone
```

1.0.2 Multiline Statements

[2]:
```
#Line Joing Using Triple Quotes
print("""Hello
    Every
         one""")
```

```
Hello
    Every
         one
```

[9]:
```
#Line Joing Using Blackslash
print("Hello \
Every \
one")
```

```
Hello Every one
```

1.0.3 Comments

[7]:
```
#Single Line Comment

''' In Python
multi line
comments
can be done
```

```
by using
triple quotes'''

#Multiple Line
#Comments
a = 10
a
```

[7]:
```
10
```

1.0.4 Indentation

[11]:
```
if 5>2:      #using if statement to check graeter number
    print("5 Is Greater")
else:
    print("2 Is Greater")
    print("Error")
```

```
5 Is Greater
```

1.0.5 Multiple Statements in single line

[12]:
```
a=10 ; print(a)
```

```
10
```

1.0.6 Quotations ' '' """

[4]:
```
print('Hello')
#Blanklines
print("Hello")

print('''Hello''')

print("""Hello""")
```

```
Hello
Hello
Hello
Hello
```

[15]:
```
#Quotes in between quotes
para = "i'm nitheesh"
para2 = 'studying "Mechanical"'
print(para)
```

```
i'm nitheesh
```

1.1 Variables

[17]:
```
#Assigning - Dynamically Typed
a=10
a
```

[17]:
```
10
```

[2]:
```
#Reassigning
b = "k"
c = b
print(c)
```

```
k
```

[16]:
```
#Multiple Values to Multiple Variables

a,b,c=20,10,30

print(a)
print(b)
print(c)
```

```
20
10
30
```

[18]:
```
#Single Values to Multiple Variables

d=e=f=10

print(d)
print(e)
print(f)
```

```
10
10
10
```

1.1.1 Swapping of varaibles

[19]:
```
a=10
b=20

print(a)
print(b)
```

```
10
20
```

[21]:
```
a,b=b,a
```

[22]:
```
print(a)
print(b)
```

```
20
10
```

1.1.2 To-Do : Write a python program to swap variables using temp variable

1.1.3 Delting Variables

[23]:
```
a=20
print(a)
```

```
20
```

[24]:
```
del a
print(a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-c342333e421e> in <module>
      1 del a
----> 2 print(a)

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

1.1.4 Rules For Variable Names

[21]:
```
#Rules For Variable Names

#1a=10

_a_=10 #First letter has to be _ or Alphabet

variableValue=20 #Can Have Numbers After First Letter

#a @=10 #No Special Characters , space Allowed

print(variableValue) #Case_Senstive
```

```
20
```

1.1.5 Keywords

[27]:
```python
import keyword
print (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']
```

[28]:
```python
s="continue"
print(keyword.iskeyword(s))
```

```
True
```

[29]:
```python
s="Python Parasites"
print(keyword.iskeyword(s))
```

```
False
```

2 Zen of Python

[30]:
```python
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!
```