Python for DevOps

Module 2.

Data types

Data types

Type	Description
Bool	Boolean value
Int	Integer
Float	Floating-point number
Tuple	Immutable sequence of objects
Str	Character string
Frozenset	Immutable form of set class
List	Mutable seqence of objects
Set	Unordered set of distict objects
Dict	Associative mapping

What can we do with data?

Assignment:

```
a = 123
```

Multiple Assignment:

```
a = b = c = 123
a, b, c, = “letter A”, “letter C”, “letter C”

a, b, *names = 1, 2, 'Jhon', ‘Jane', 'Mike'
print(names)
['Jhon', 'Jane', 'Mike']
print(type(names))
<class 'list'>
```

Deletion:

```
del var1
del var_a, var_b
```

Implicit Type Conversion

```
x = 10
y = 10.6
z = x + y
print("z is of type:",type(z))
z is of type: <class 'float'>
```

Explicit Type Conversion

```
int(x [base]), float(x),
str(x), chr(x),
ord(x), hex(x),
oct(x), bin(x)
```

Operators in Python

Operators could be:

Arithmetic

Comparison

Assignment

Logical

Bitwise

Membership

Identity

Arithmetic Operators

Operator	Action
+	Addition
-	Subtraction
*	Multiplication
/	Division (float)
%	Modulus
**	Power
//	Division (floor)

Comparison Operators

Operator	Action
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Assignment Operators

Operator	Action	Syntax
=	Assign value	x = y + z
+=	Add AND	a+=b a = a + b
-=	Substract AND	a-=b a = a - b
*=	Multiply AND	a*=b a = a * b
/=	Divide AND	a/=b a = a / b
%=	Modulus AND	a%=b a = a % b
//=	Divide(floor) AND	a//=b a = a // b

Logical Operators

Operator	Action
and	Logical AND
or	Logical OR
not	Logical NOT

Bitwise Operators

Operator	Action
&	Bitwise AND
|	Bitwise OR
~	Bitwise NOT
^	Bitwise XOR
>>	Bitwise right shift
<<	Bitwise left shift

Membership Operators

Operator	Action
in	True if value is found in the sequence
not in	True if value is not found in the sequence

Identity Operators

Operator	Action
is	True if the operands are identical
is not	True if the operands are not identical

Priority of operators

Conditions

If … else …

Full form

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Short Hand

```
if a > b: print("a is greater than b")

a = 2
b = 330
print("A") if a > b else print("B")
```

Nested If

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

The pass Statement

```
a = 33
b = 200

if b > a:
  pass
```

Loops

“for” and “while” loops

```
numbers = [1, 2, 3, 4, 5]
for number in numbers:
    print(number)
```

```
1
2
3
4
5
```

```
for x in range(3):
    print(x)
```

```
0
1
2
```

```
for x in range(3, 8, 2):
    print(x)
```

```
3
5
7
```

```
number = 0
while number < 5:
    print(number)
    number+=1
```

```
0
1
2
3
4
```

"break" and "continue" statements

```
number = 0
while True:
    print(number)
    number+=1
    if number >=5:
        break
```

```
0
1
2
3
4
```

```
for x in range(10):
    if x % 2 == 0:
        continue
    print(x)
```

```
1
3
5
7
9
```

```
number = 0
while number < 5:
    print(number)
    number+=1
else:
    print(f"number reached {number}")
```

```
0
1
2
3
4
number reached 5
```

* else could be ignored if loop terminated because of “break” but not due to fail in condition

Numbers

Numbers

int	10	1 (int) + 1.2 (float) = 2.2 (float)
float	1.0	1 (int) + 2j (complex) = 1+2j (complex)
complex	3.14j	1+2j (complex) + 2 (int) = 3+2j (complex)

decimal 1.0

```
>>> (1.1 + 2.2) == 3.3
False
```

```
import decimal
```

Complex mathematics

https://docs.python.org/3/library/math.html

Import math

	Output
print(math.pi)	3.141592653589793
print(math.cos(math.pi))	-1.0
print(math.exp(10))	22026.465794806718
print(math.log10(1000))	3.0
print(math.factorial(6))	720

Pseudorandom number generator

```
import random
print(random.randrange(1,100))
34

# Winner picker:
names = ['Mike', 'Alice', 'Jhon', ‘Vasya']
print(random.choice(names))
Alice


#Music collection shuffle:
tracks = ['1.mp3', '2.mp3', '4.wav']
random.shuffle(tracks)
print(tracks)
['4.wav', '1.mp3', '2.mp3']
```

Strings

Strings

```
expression = "I am Groot”
print(expression)

print(expression[0])
I

for character in expression:
    print(character)
```

Let’s cut the string

```
expression = "I am Groot"
print(expression[0:4])
I am
print(expression[5:])
Groot
print(expression[::-1])
toorG ma I
```

```
a[start:stop]   # items start through stop-1
a[start:]       # items start through the rest of the array
a[:stop]        # items from the beginning through stop-1
a[:]            # a copy of the whole array
```

String Formatting: “Old style”

https://docs.python.org/3/library/stdtypes.html#old-string-formatting

String Formatting (% Operator)

```
name = "Nick"
'Hello, %s' %name
'Hello, Nick’
```

```
name = 'Nick’
friend = 'Bob'
'Hi %(uname)s, this is your friend %(fname)s!' % {"uname": name, "fname": friend}
'Hi Nick, this is your friend Bob!'
```

“New Style” String Formatting (str.format)

```
name = 'Nick'
'Hello {},'.format(name)
'Hello Nick,'


'Hi {}, here is your friend {}'.format(name, friend)
'Hi Nick, here is your friend Bob'

'Hi {uname}, here is your friend {fname}'.format(fname=friend, uname=name)
'Hi Bob, here is your friend Nick'
```

String Interpolation / f-Strings

```
name = 'Nick'
f'Hello, {name}'
'Hello, Nick'
```

```
a = 2
b = 5
>>> f"it's math time. a+b*a {a+b*a} is not the same as (a+b)*a {(a+b)*a}"
"it's math time. a+b*a 12 is not the same as (a+b)*a 14"
```