Oracle Cloud Infrastructure

Study Guide

Infrastructure as Code

Release 04/2018

ORACLE Certified Associate

Walter Goerner

Version: 04/18

Walter Goerner
Director Cloud Business Development Oracle EMEA

ORACLE

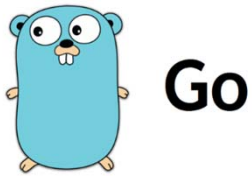
OCI Software Development Kit

The OCI Software Development Kit

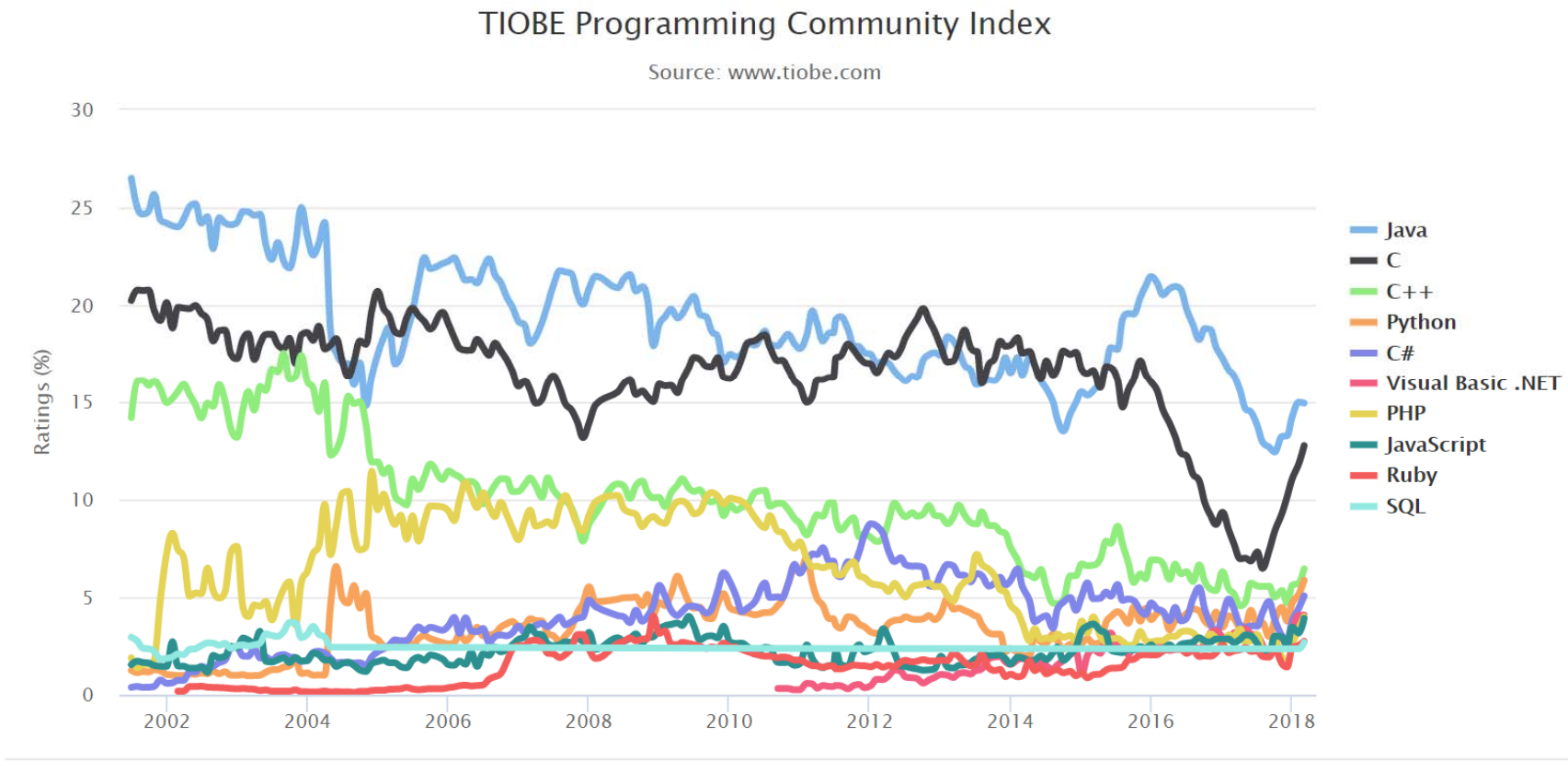
In this chapter we will cover the following topics:

- OCI SDK overview
- Overview of the Python programming language
- Python SDK installation and configuration
- Basic Python SDK building blocks and examples
- Using Visual Studio Code as a Python IDE
- Additional Python SDK Examples
- Advanced Python SDK Building Blocks

Supported Programming Languages/Environments

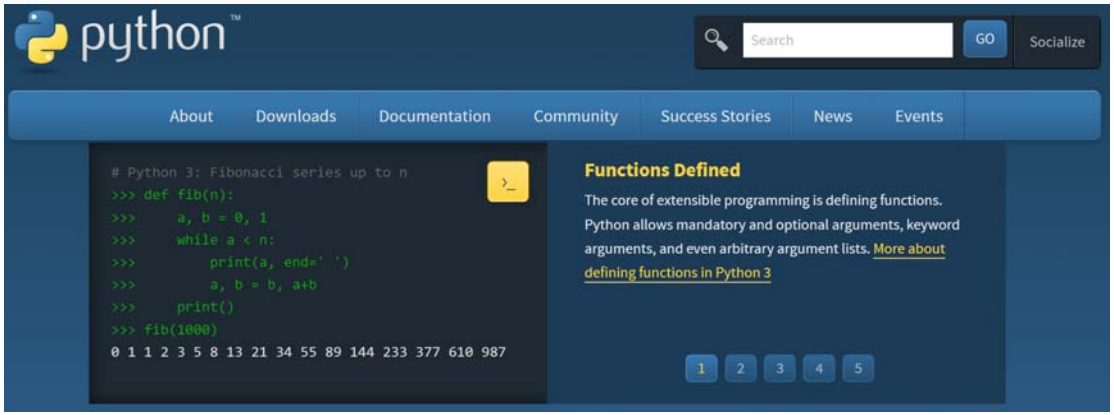


Python Overview (1)



Python Overview (2)

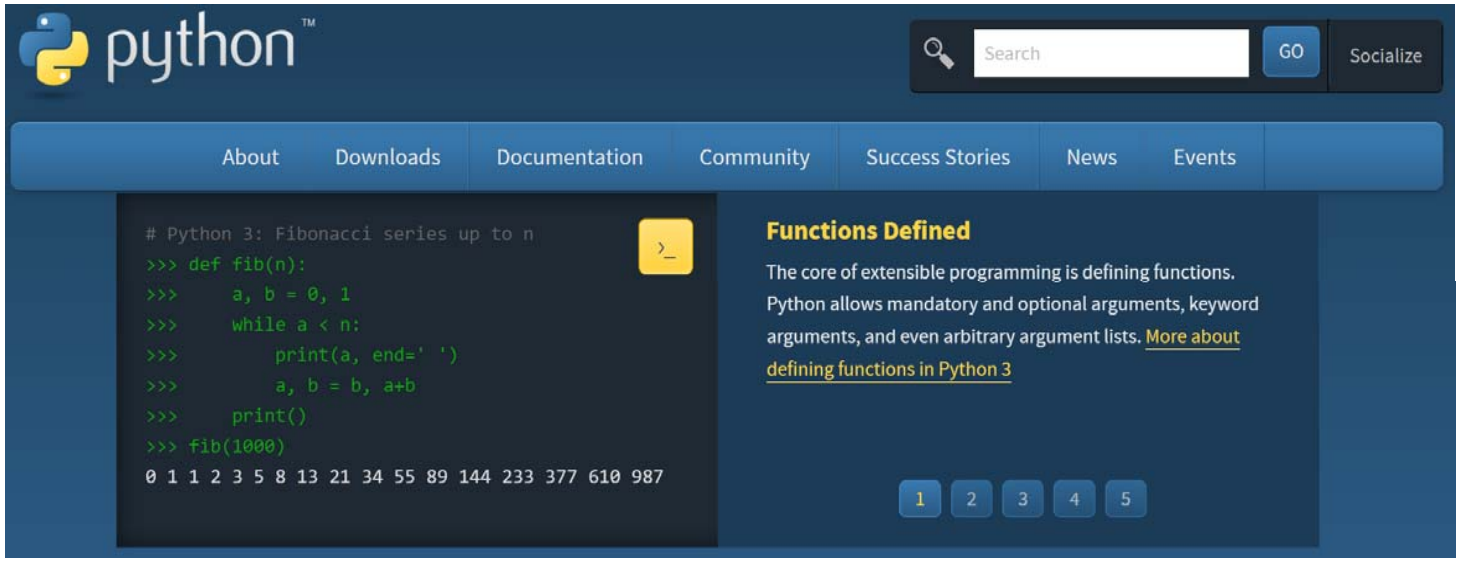
Mar 2018	Mar 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.941%	-1.44%
2	2		C	12.760%	+5.02%
3	3		C++	6.452%	+1.27%
4	5	▲	Python	5.869%	+1.95%
5	4	▼	C#	5.067%	+0.66%



Why Python?

- **Less application development time:** Python code is usually 2 – 10 times shorter than comparable code written in languages like C/C++ and Java, which means that you spend less time writing your application.
- **Ease of reading:** A programming language is like any other language — you need to be able to read it to understand what it does. Python code tends to be easier to read than the code written in other languages, which means you spend less time interpreting it.
- **Reduced learning time:** The creators of Python wanted to implement a programming language with fewer odd rules that make the language hard to learn. After all, we want to create OCI scripts, not learn obscure and difficult programming languages.

Python Software Foundation home page www.python.org



https://github.com/oracle/oci-python-sdk

When looking for documentation on the OCI Python SDK your start point will be the official Python SDK homepage at

https://oracle-cloud-infrastructure-python-sdk.readthedocs.io/en/latest/installation.html

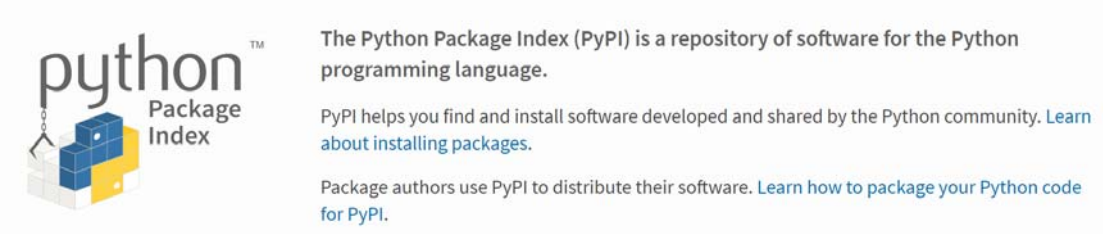
Python Resources

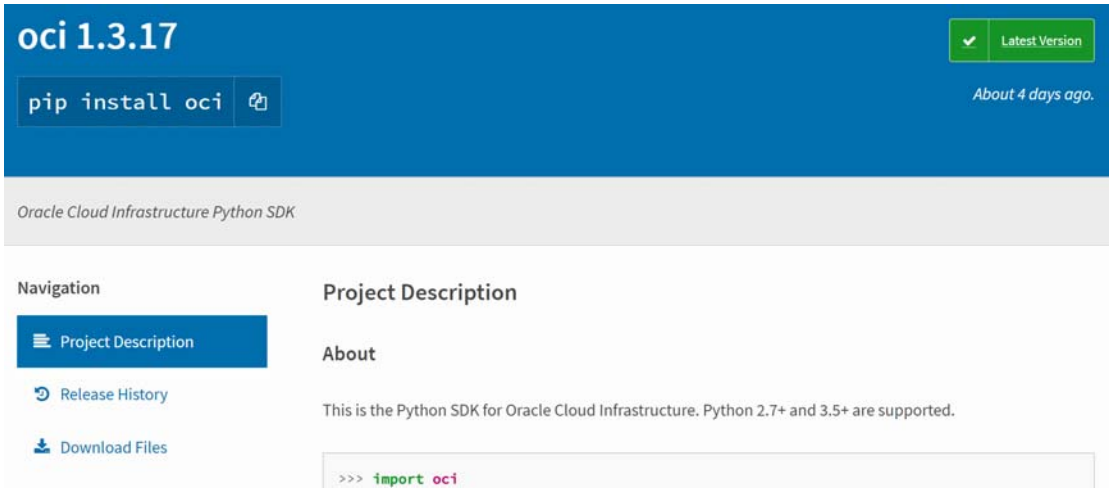
Resource	Link
Python Beginner's Guide for non-programmers	https://wiki.python.org/moin/BeginnersGuide/NonProgrammers
Popular Python Recipes	http://code.activestate.com/recipes/langs/python/
The Python Tutorial	https://docs.python.org/3/tutorial/
Intro to Python	https://overiq.com/python/3.4/intro-to-python/
The Best Python Books	https://pythonbooks.org/

SDK Installation Requirements

Requirement	Description
	An **Oracle Cloud Infrastructure account**, i.e. a tenancy together with a compartment.
	An **OCI user** created in that account, in a group with a policy that grants the desired permissions for the OCI resources to be managed. This account user can be or yourself, another person, or a system that calls the API.
	A **keypair** used for signing API requests, with the public key uploaded to Oracle. Only the user calling the API should possess the private key.
	**Python** version 2.7.5 or 3.5 or later, running on Mac, Windows, or Linux. Note that the Python SDK uses the **`cryptography.io`** library, which has its own build requirements. Ensure that you satisfy those requirements prior to installing the Python SDK.

PyPi, the Python Package Index - https://pypi.org





Preferred Installer Program - PIP

```
PS C:\Users\Tux> pip

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  search                      Search PyPI for packages.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  help                        Show help for commands.
[…]
```

```
pip install <package name>
```

and

```
pip install <package name> --upgrade
```

Prerequisite - **cryptography.io**

Coming back to installing the Python OCI SDK one of the package requirements is for the **`cryptography.io`** package which can be installed using pip before the actual SDK installation

```
PS C:\Windows\system32> pip install cryptography
```

or

```
[tux@oraclelinux ~]$ sudo pip install cryptography
```

SDK Installation

```
PS C:\Windows\system32> pip install oci
Collecting oci
  Downloading oci-1.3.15-py2.py3-none-any.whl (647kB)
    100% |████████████████████████████████| 655kB 1.3MB/s
Collecting python-dateutil==2.5.3 (from oci)
  Downloading python_dateutil-2.5.3-py2.py3-none-any.whl (201kB)
    100% |████████████████████████████████| 204kB 1.9MB/s
Collecting pyOpenSSL<=17.4.0 (from oci)
  Downloading pyOpenSSL-17.4.0-py2.py3-none-any.whl (52kB)
[…]
Successfully installed PyJWT-1.5.3 certifi-2018.1.18 chardet-3.0.4 configparser-3.5.0
cryptography-2.1.3 httpsig-cffi-15.0.0 oci-1.3.15 pyOpenSSL-17.4.0 python-dateutil-2.5.3
pytz-2016.10 requests-2.18.4 urllib3-1.22
```

The configuration of the Python SDK is done in exactly the same way as for the OCI CLI, i.e. the Python SDK makes use of an already existing `~/.oci/config` file with all the necessary configuration parameters already described when configuring the OCI CLI. It's thus fair to assume that if the OCI CLI works fine on your computer, the Python SDK will work fine as well.

A First Python SDK Example

Let's test this with a simple Python script using the OCI SDK **(2_first_test.py)**:

```
PS C:\Users\Tux> python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.

>>> import oci
>>> config = oci.config.from_file("~/.oci/config")
>>> identity = oci.identity.IdentityClient(config)
>>> user = identity.get_user(config["user"]).data
>>> print(user)
```

Basic Building Blocks and Examples **(3_basics.py)** (1)

```
>>> import oci
>>> config = oci.config.from_file()
```

```
>>> config
{'log_requests': False, 'additional_user_agent': '', 'pass_phrase': None, 'user':
'ocid1.user.oc1..aaaaaaaaizoxjynjijsvkhzxe3yaw67g5fvjxontbxv2xokvexneuwdsfo7a',
'fingerprint': '42:22:b1:f8:a0:66:5c:3c:91:02:57:56:32:80:04:e9', 'key_file':
'C:\\Users\\Tux\\.oci\\oci_api_key.pem', 'tenancy':
'ocid1.tenancy.oc1..aaaaaaaaa75b4lz3hvodfw67q3d7dzrgnryogxlrnwjnljqxle63vdlt777q',
'region': 'eu-frankfurt-1'}
```

```
>>> identity = oci.identity.IdentityClient(config)
>>> compartment_id = config["tenancy"]
```

```
>>> identity.base_client.endpoint
'https://identity.eu-frankfurt-1.oraclecloud.com/20160918'
```

Basic Building Blocks and Examples **(3_basics.py)** (2)

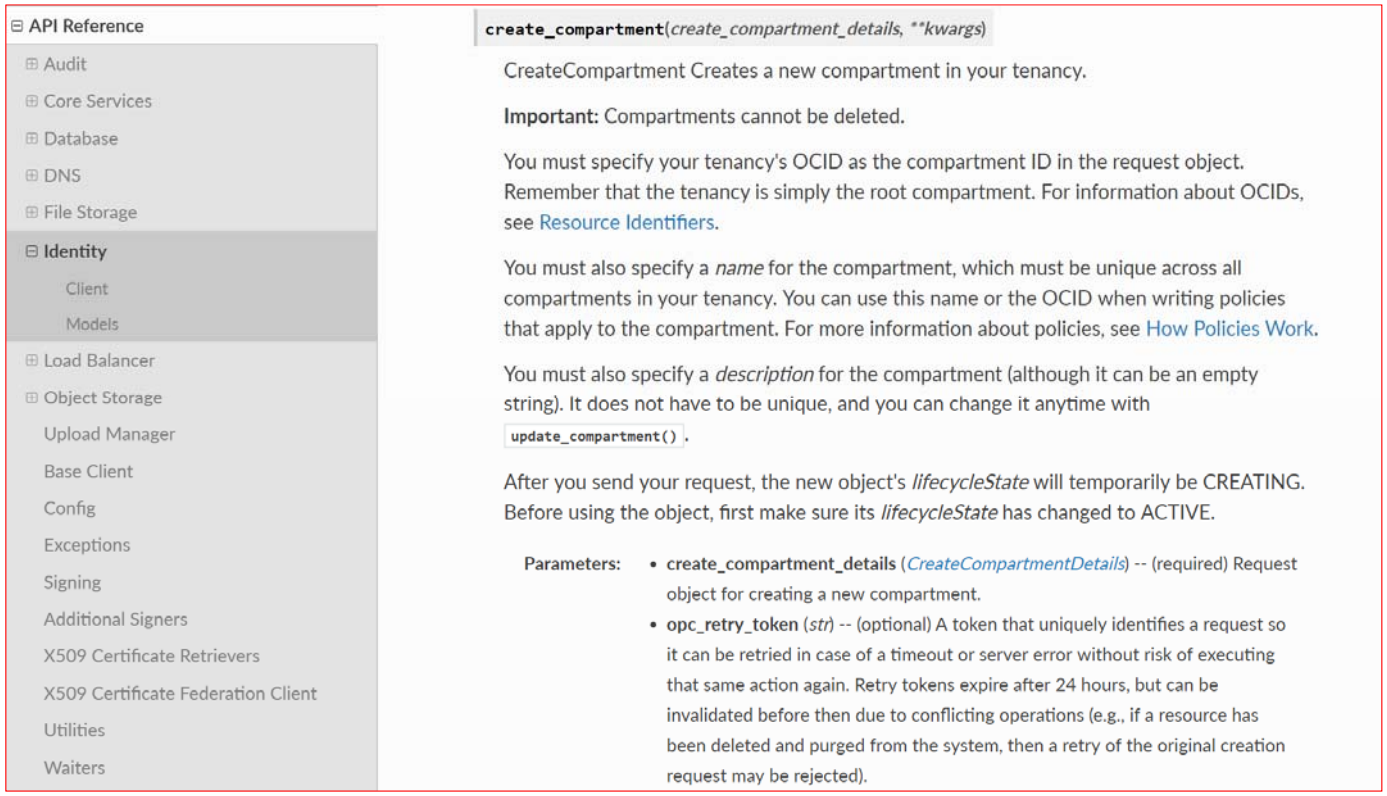
```
>>> regions=identity.list_regions()
>>> regions.data
[{
  "key": "FRA",
  "name": "eu-frankfurt-1"
}, {
  "key": "IAD",
  "name": "us-ashburn-1"
```

```
>>> compartment_id = config["tenancy"]
>>> compartment_id
'ocid1.tenancy.oc1..aaaaaaaaa75b4lz3hvodfw67q3d7dzrgnryogxlrnwjnljqxle63vdlt777q'
```

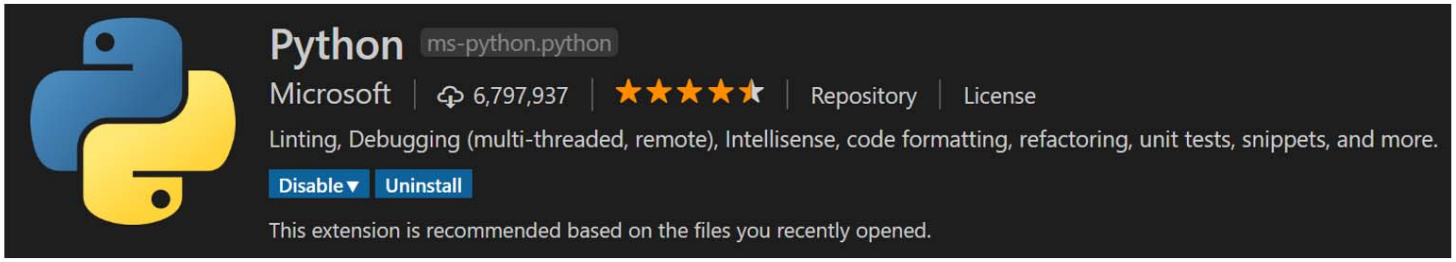
```
>>> userlist = identity.list_users(compartment_id)
>>> userlist.data
[{
  "compartment_id": "ocid1.tenancy.oc1..aaaaaaaaa75b4lz3hvodfw67q3d7dzrgnryog[…]",
  "defined_tags": {},
  "description": "Walter Goerner",
```

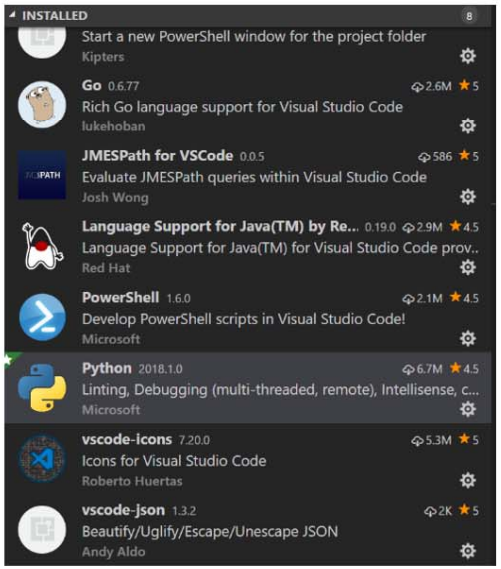
API Reference Page

https://oracle-cloud-infrastructure-python-sdk.readthedocs.io/en/latest/api/index.html



VS Code as Python IDE





The Python extension should recognize the installed version of Python automatically, if not then specify using the **Python: Select Interpreter** command on the **Command Palette** (`Ctrl+Shift+P`).

On the Microsoft page for VS Code there is a Python tutorial at

https://code.visualstudio.com/docs/python/python-tutorial

that walks you thru the various options and features of the Python extension for VS Code. Especially the tutorial on debugging is very useful for developing OCI Python scripts.

Lab 3

Installing and Configuring the Python SDK

Lab 3: Installing and Configuring the Python SDK

Objectives

The objectives of this lab are to install and configure the Python SDK on your Computer also verifying basic SDK functionality. The second main objective is to configure VS Code for Python usage and to be able to debug Python OCI SDK programs using the VS Code IDE.

Main tasks for this lab

- **Task 1:** Install and configure the Python SDK on your computer
- **Task 2:** Test basic SDK functionality using command-line Python
- **Task 3:** Configure VS Code for Python language support
- **Task 4:** Use VS Code for debugging simple Python SDK scripts

Estimated Time: 45 minutes

Additional Python SDK Examples **(4_create_user.py)**

```
1   # Creating an OCI user in Python demo script
2   import oci
3   config = oci.config.from_file()
4   identity_client = oci.identity.IdentityClient(config)
5   compartment_id = config["tenancy"]
```

The interesting line is the **service client** that is initialized in line 3; that's the connection between the Python client to a specific API service, in this case the identity service.

```
7   # prepare OCI request by assigning the desired user parameters
8   from oci.identity.models import CreateUserDetails
9   request = CreateUserDetails()
10  request.compartment_id = compartment_id
11  request.name = "python-user"
12  request.description = "Created with the Python SDK"
13  user = identity_client.create_user(request)
14  print(user.data.id)
```

Create a VCN **(5_Create_vcn.py)** (1)

```
1   # Creating a VCN in Python demo script
2   import oci
3   config = oci.config.from_file()
4   identity_client = oci.identity.IdentityClient(config)
5   tenancy_id = config["tenancy"]
6   compartment = identity_client.list_compartments(tenancy_id)
7   print(compartment.data)
```

Assign the OCID of the POCCOMP1 compartment to the `compartment_id` variable for future use:

```
9   # change the assignment below to match your POCCOMP1 OCID
10  compartment_id = "ocid1.compartment.oc1..aaaaaaaaoadg47pc2djpkgk3yckfnm[…]"
```

In the next line we need a new service client for the network API service that is responsible for managing VCNs:

```
11  virtual_network_client = oci.core.virtual_network_client.VirtualNetworkClient(config)
```

Create a VCN **(5_Create_vcn.py)** (2)

```
13  # prepare OCI request by assigning the desired VCN parameters
14  from oci.core.models import CreateVcnDetails
15  request = CreateVcnDetails()
16  request.compartment_id = compartment_id
17  request.display_name = "pythonvcn"
18  request.dns_label = "pythonvcn"
19  request.cidr_block = "172.0.0.0/16"
```

The last step is to call the `create_vcn` function and assign the resulting object to a variable `vcn`:

```
21  vcn = virtual_network_client.create_vcn(request)
```

User and Group Management **(6_user_crud.py)** (1)

```
4  import oci
5  from oci.identity.models import AddUserToGroupDetails, CreateGroupDetails,
   CreateUserDetails
6
7  # Default config file and profile
8  config = oci.config.from_file()
9  compartment_id = config["tenancy"]
10
11 # Service client
12 identity = oci.identity.IdentityClient(config)
13
```

User and Group Management **(6_user_crud.py)** (2)

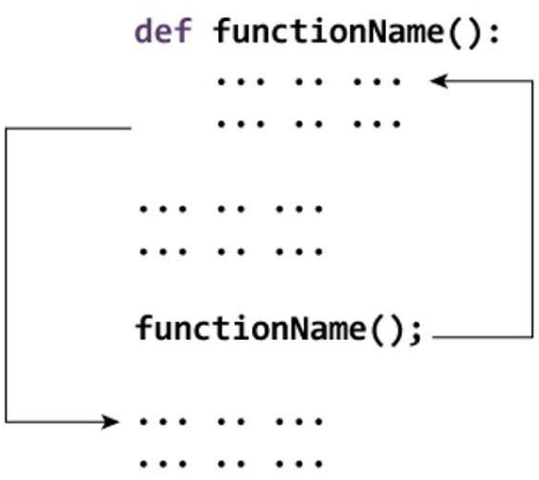
```
14 user_name = "python-sdk-example-user"
15 group_name = "python-sdk-example-group"
16
17 print("Creating a new user {!r} in compartment {!r}".format(user_name, compartment_id))
```

The user and group names are assigned in lines 14 and 15 and printed out in line 17 as a raw string.

```
19 request = CreateUserDetails()
20 request.compartment_id = compartment_id
21 request.name = user_name
22 request.description = "Created by a Python SDK example"
23 user = identity.create_user(request)
24 print(user.data)
```

Additional Python Constructs: Functions

A **function** is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. Here are some simple rules to define a function in Python:



- Function blocks begin with the keyword **def** followed by the function name and parentheses.
- Any input parameters or arguments should be placed within these parentheses.
- The code block within every function starts with a colon **:** and is indented.
- The statement **return** [expression] exits a function, optionally passing back an expression to the caller.

Additional Python Constructs: Error Handling

Python is implementing **error handling** via structured exception processing using the **try** statement. A critical operation which can raise an exception is placed inside the try clause and the code that handles the exception is written in the **except** clause. Look at the following code snippet to the left below and the output to the right to illustrate the concept:

```
1   # import module sys to get the type of exception
2   import sys
3
4   randomList = ['a', 0, 2]
5
6   for entry in randomList:
7       try:
8           print("The entry is", entry)
9           r = 1/int(entry)
10          break
11      except:
12          print("Oops!",sys.exc_info()[0],"occured.")
13          print("Next entry.")
14          print()
15  print("The reciprocal of",entry,"is",r)
```

```
The entry is a
Oops! <class 'ValueError'> occured.
Next entry.

The entry is 0
Oops! <class 'ZeroDivisionError' > occured.
Next entry.

The entry is 2
The reciprocal of 2 is 0.5
```

Python Function for Creating a VCN (1)

```
1  # Python function to create a VCN
2
3  import oci
4
5  def create_vcn(virtual_network, compartment_id, vcn_name, dns_label, cidr_block):
6      result = virtual_network.create_vcn(
7          oci.core.models.CreateVcnDetails(
8              cidr_block=cidr_block,
9              display_name=vcn_name,
10             compartment_id=compartment_id,
11             dns_label=dns_label
12         )
13     )
14     get_vcn_response = oci.wait_until(
15         virtual_network,
16         virtual_network.get_vcn(result.data.id),
17         'lifecycle_state',
18         'AVAILABLE'
19     )
20     print('Created VCN: {}'.format(get_vcn_response.data.id))
21
22     return get_vcn_response.data
23
```

Python Function for Creating a VCN (2)

```
24 config = oci.config.from_file()
25 virtual_network_client = oci.core.VirtualNetworkClient(config)
26 compartment_id = config["tenancy"]
27
28 try:
29     vcn = create_vcn(virtual_network_client, compartment_id, "pythonvcn1",
   "pythonvcn1", "172.0.0.0/16")
30     print(vcn)
31
32 finally:
33     exit()
```

Lab 4

Using the Python SDK to Provision a POC Environment

Lab 4: Using the Python SDK to Provision a POC Environment

Objectives

The objective of this lab is to leverage the Python SDK to provision, setup, and configure another Proof of Concept (POC) environment running in OCI for OCI database services POCs. The lab is considered complete when you are able to connect to the database service successfully. You are made aware that there is a sample Python script on the Python SDK GitHub site that will help you get started. At the end of this lab all the resources provisioned should be destroyed.

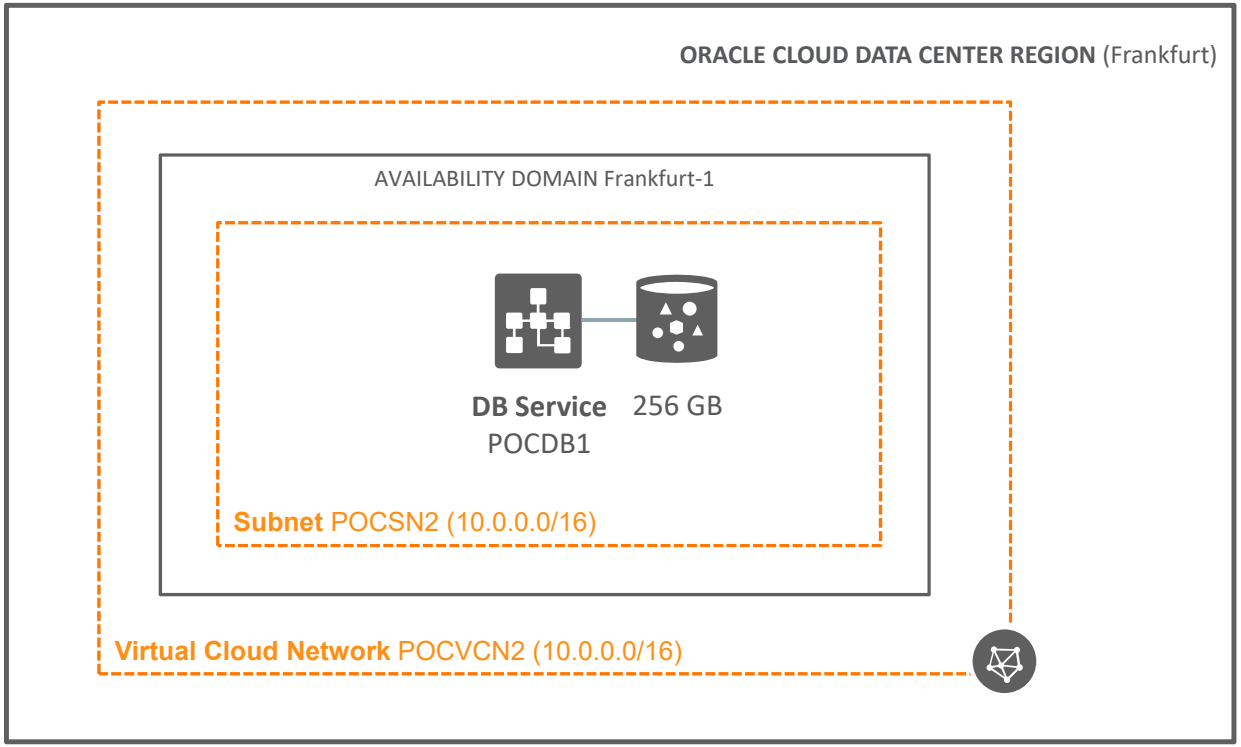
Main tasks for this lab

- **Task 1:** Review and analyze the desired OCI DB POC architecture
- **Task 2:** Analyze and understand the sample Python script on GitHub
- **Task 3:** Adapt the sample Python script to generate the desired POC architecture
- **Task 4:** Debug and test your Python script

Estimated Time: 60 minutes

Lab 4: DB POC Environment

Compartment POCCOMP1



DB Service
POCDB1

DB-Edition: Enterprise
DB-Shape: VM.Standard1.2
DB-Name: DB1
DB-Version: 12.2.0.1
DB-Size: 256 GB
RSA Public/Private Key Pair
DB Admin Password: -

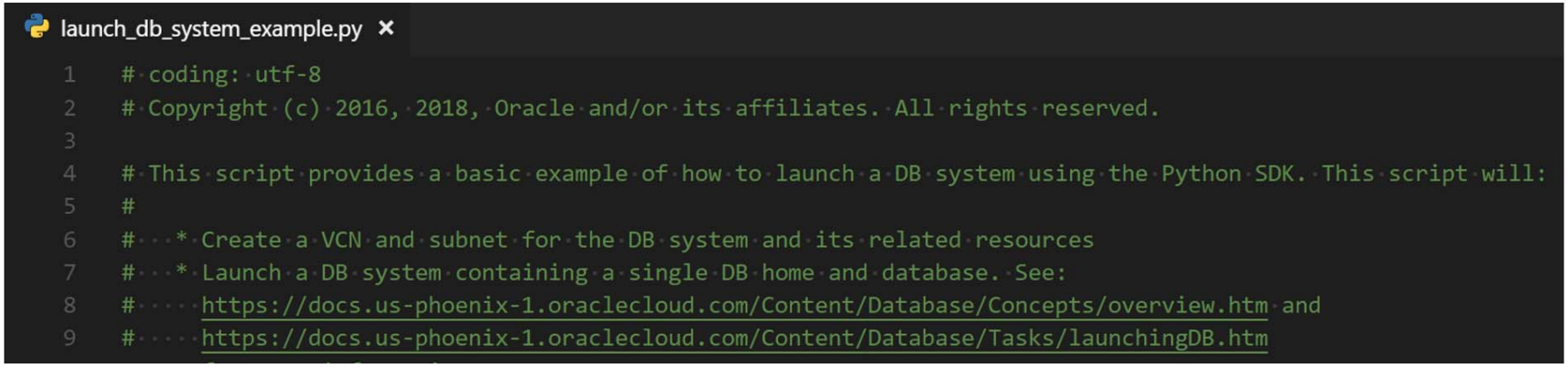
Lab 4 Guidance (1)

➢ **Task 2: Analyze and understand the sample Python script on GitHub**

The sample DB Python script can be found on GitHub:

https://github.com/oracle/oci-python-sdk/blob/master/examples/launch_db_system_example.py

Download the file and open the Python script in Visual Studio Code:



Lab 4 Guidance (2)

Use Visual Studio Code and its debugging/tracing capabilities to step thru the script analyzing and understanding the functionality. The sample script is designed to accept four parameters as command line arguments (`compartment_id`, `availability_domain`, `cidr_block`, and `ssh_public_key_path`). Although this is generally a good practice and provides flexibility to adapt the script to various OCI environments, it makes debugging the script more complex (although this could also be accomplished in VS Code).

To change the default behavior of the script, look at the following code lines:

```
192  if len(sys.argv) != 5:
193      raise RuntimeError('Invalid number of arguments provided to the script. Consult the script header for required arguments')
194
195  compartment_id = sys.argv[1]
196  availability_domain = sys.argv[2]
197  cidr_block = sys.argv[3]
198  ssh_public_key_path = os.path.expandvars(os.path.expanduser(sys.argv[4]))
```

The Python command line parameters are passed to the script in the `argv` array and line 192 checks whether the number of arguments passed is not equal to 5 (counting starts at 0). If so, a runtime error is raised in line 193 and the script is terminated.

Lab 4 Guidance (3)

To change this behavior simply delete lines 192 and 193 and move lines 195 to 198 up to the top of the script to line 40 changing the variable assignments to “hard code” the required data:

```
34  ADMIN_PASSWORD = "ADummyPassw0rd_#1"
35  DB_VERSION = '12.1.0.2'
36  DB_SYSTEM_CPU_CORE_COUNT = 4
37  DB_SYSTEM_DB_EDITION = 'ENTERPRISE_EDITION'
38  DB_SYSTEM_SHAPE = 'BM.DenseIO1.36'
39
40  compartment_id = "<OCID of compartment>"
41  availability_domain = "<Name of avavailability domain>"
42  cidr_block = "<IP Block to be used>"
43  ssh_public_key_path = "<path to public key file>"
```

Enter the actual variables to be used for your OCI environment and you should be good to go for testing the script functionality.

Review Questions

OCI Python Software Development Kit

Review Questions for OCI Python SDK (1)

1. Oracle supports which of the following programming languages for the OCI SDK?
   - **A.** Java
   - **B.** Python
   - **C.** C++
   - **D.** Go

2. Which of the following are prerequisites for installing the OCI SDK?
   - **A.** An Oracle Developer Network account
   - **B.** Python 2.7.5 or 3.5 or later installed on your computer
   - **C.** A public/private keypair for signing API requests
   - **D.** Visual Studio Code installed on your computer

3. What is the command to install the Python OCI SDK?
   - **A.** `yum install python-oci-sdk`
   - **B.** `nuget python/oci`
   - **C.** `pip install oci`
   - **D.** `apt-get oci_sdk`

Review Questions for OCI Python SDK (2)

4. Which of the following statements regarding the OCI SDK is true?

A. Oracle supports the OCI SDK only for Java.

B. The OCI SDK is only available to existing Oracle Support customers.

C. The OCI SDK can be downloaded from GitHub.

D. The OCI SDK only works on computers running Oracle Linux.

5. Which of the following statements regarding the OCI SDK are true?

A. The OCI SDK cannot be installed on a computer with the OCI CLI already installed.

B. The OCI SDK can use the same config file as the OCI CLI tool.

C. Visual Studio Code provides support for the OCI SDK out of the box.

D. Visual Studio Code can be configured as a Python IDE.

Review Questions for OCI Python SDK (3)

6. How is a function defined in Python?
   - **A.** A function is defined with the keyword `function`, the function statements need to enclosed using {}.
   - **B.** A function is defined with the keyword `fun`, the function statements need to enclosed using {}.
   - **C.** A function is defined with the keyword `fun`, the function statements need to be grouped by indenting.
   - **D.** A function is defined with the keyword `function`, the function statements need to be grouped by indenting.

7. Which of the following Python statements are needed for a default basic OCI SDK configuration?
   - **A.** `config = oci.config.from_file()`
   - **B.** `import sys`
   - **C.** `import oci`
   - **D.** `virtual_network_client = oci.core.VirtualNetworkClient(config)`

8. How is structured exception handling implemented in Python?
   - **A.** Use the `error_handler` function.
   - **B.** Use the `try` and `finally` keywords.
   - **C.** Use the `catch` and `throw` keywords.
   - **D.** There is no structured exception handling in Python.

Review Questions for OCI Python SDK (4)

9. How can you programmatically wait for an OCI resource to become available in Python?

A. Keep re-creating the resource until it is available.

B. Insert a wait loop for 5 minutes after creating any OCI resource.

C. Use the `oci.wait_until()` function.

D. There is no way of doing this programmatically, you have to look in the web console.

10. Which of the following Python expressions create a service client for OCI assuming that the variable config is correctly initialized using `oci.config.from_file`?

A. `database_client = oci.database.DatabaseClient(config)`

B. `virtual_network = oci.core.VirtualNetworkClient(config)`

C. `identity = oci.identity.IdentityClient(config)`

D. All of the above.


рахмат
danke
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
mersi
barka
welalin
tack
spas
dank je
misaotra
paldies
grazzi
mahalo
tapadh leat
vinaka
спасибі
blagodaram
matondo
kiitos
dankie
nanni
nandri
dhanyavad
thank you
gracias
хвала
asante
manana
obrigada
tenki
bedankt
bayarlalaa
gracie
hvala
mauruuru
köszönöm
dankon
akun
aciú
enkosi
dziękuję
djiere dieuf
tau
mochchakkeram
murakoze
дякую
mamnun
chnorakaloutioun
gratias ago
gràcies
sulpay
go raibh maith agat
chokrane
sobodi
dėkuji
sagolun
sukriya
kop khun krap
taiku
arigatō
takk
dakujem
trugarez
obrigado
mesi
didi madloba
ありがとう
najis tuke
tanemirt
grazie
rahmet
kam sah hamnida
rahmat
terima kasih
diolch
dhanyavadagalu
shukriya
merce
мерси
তোমাকে ধন্যবাদ
감사합니다
xiexie
ευχαριστώ
merci

ORACLE

Integrated Cloud

Applications & Platform Services


ORACLE®