KCL Language

Cheat sheet
Version (latest)

Intro DOC

KCL is an opensource constraint-based record & functional language mainly used in configuration and policy scenarios.

https://github.com/KusionStack/KCLVM
https://kcl-lang.io

Installation DOC

- **Docker**

```
docker pull kusionstack/kclvm
```

- **macOS**

```
brew install kcl-lang/tap/kclvm
```

- **Linux**

```
wget -q https://kcl-lang.io/script/install.sh -O
      - | /bin/bash
```

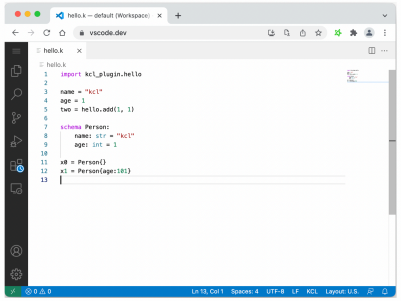
- **Windows**

```
powershell -Command "iwr -useb https://kcl-lang.
      io/script/install.ps1 | iex"
```

Quick start DOC

```
# This is a KCL document

title = "KCL Example"

owner = {
    name = "The KCL Authors"
    data = "2020-01-02T03:04:05"
}

database = {
    enabled = True
    ports = [8000, 8001, 8002]
    data = [["delta", "phi"], [3.14]]
    temp_targets = {cpu = 79.5, case = 72.0}
}

servers = [
    {ip = "10.0.0.1", role = "frontend"}
    {ip = "10.0.0.2", role = "backend"}
]
```

VS Code Extension LINK



Highlighting, auto-completion, quick info hover and code navigation, etc

Keywords DOC

```
True    False  None      Undefined  import
and     or     in        is         not
as      if     else      elif       for
schema  mixin  protocol  check      assert
all     any    map       filter     lambda
rule
```

Data Types(bool/int/float/units/str) DOC

```
name = "Foo"  # exported, can only be set once.
_name = "Foo"  # internal and are mutable
$if = 3  # equal to `if = 2`
```

boolean

```
a = True
b = False
```

int

```
a = 123
b = 0x10 # hexadecimal literal
c = int("10") # int constructor
```

float

```
a = 1.10
b = -35.59
c = 32.3e+18
d = 70.2E-12
e = float("112") # float constructor
```

units

```
# SI
n = 1n  # 1e-09
u = 1u  # 1e-06
m = 1m  # 1e-03
k = 1k  # 1000
K = 1K  # 1000
M = 1M  # 1000000
G = 1G  # 1000000000
T = 1T  # 100000000000
P = 1P  # 1000000000000000
# IEC
Ki = 1Ki  # 1024
Mi = 1Mi  # 1024 ** 2
Gi = 1Gi  # 1024 ** 3
Ti = 1Ti  # 1024 ** 4
Pi = 1Pi  # 1024 ** 5
```

str

```
'allows embedded "double" quotes'
"allows embedded 'single' quotes"

'''Three single quotes'''
"""Three double quotes"""

"""This is a long triple quoted string
may span multiple lines.
"""

s = "Hi\nHello"

# This is a KCL raw string with the `r` prefix.
raw_s = r"Hi\nHello"


x = 'The + operator ' + 'works, as well.'
x = str(3.5) # "3.5"

worldString = "world"
s = "Hello ${worldString}"

# This is a KCL raw string with the `r` prefix.
raw_s = r"Hello ${worldString}"

x = "length"
assert len(x) == 6 # True
assert "{} {}".format("ab", "12") == 'ab 12'
```

Data Types(List/Dict/Schema/Alias) DOC

list

```
list = [1, 2, 3]
assert len(list) == 3  # True
assert list[0] == 1  # True

list = [ _x for _x in range(10) if _x % 2 == 0]
assert list == [0, 2, 4, 6, 8] # True

# [1000, 2000, 3000]
dataLoop = [i if i > 2 else i + 1 for i in data]
```

dict

```
a = {"one" = 1, "two" = 2, "three" = 3}
b = {'one' = 1, 'two' = 2, 'three' = 3}
assert a == b # True
assert len(a) == 3 # True

person = {
    base.count = 2
    base.value = "value"
    labels.key = base.value
}
```

Schema

```
schema Person:
    firstName: str
    lastName: str
    age: int = 0 # default value: 0
```

Type Alias

```
type Int = int
type String = str
type StringOrInt = String | Int
type IntList = [int]
type StringAnyDict = {str:}
```

Operators

```
+   -   *   **  /   //  %
<<  >>  &   |   ^   <   >
~   <=  >=  ==  !=  @   \
```

Arithmetic

```
assert 2 + 3 == 5
assert 2 - 3 == -1
assert 2 * 3 == 6
assert 5 / 2 == 2.5
assert 5 // 2 == 2
assert 5 % 2 == 1
```

Equality and Relational

```
assert 2 == 2
assert 2 != 3
assert 3 > 2
assert 2 < 3
assert 3 >= 3
assert 2 <= 3
```

Bitwise and Shift

```
value = 0x22
bitmask = 0x0f

assert (value & bitmask) == 0x02
assert (value & ~bitmask) == 0x20
assert (value | bitmask) == 0x2f
assert (value ^ bitmask) == 0x2d
assert (value << 4) == 0x220
assert (value >> 4) == 0x02
```

Bitwise and Shift

```
_x = True and (col == 0 or col == 3)
```

Operators

```
empty_String = ""
empty_String is not None # True

1 in [1, 2, 3]                    # True

d = {one = 1, two = 2}
"one" in d                        # True
"three" in d                      # False
1 in d                            # False
[] in d                           # False

"nasty" in "dynasty"              # True
"a" in "banana"                   # True
"f" not in "way"                  # True

# Data is a schema with attributes one and two
d = Data {one = 1, two = 2}

"one" in d                        # True
"three" in d                      # False
```

Control Flow Statements

If

```
a = 10
if a == 0:
    print("a is zero")
elif a < 100:
    print("a < 100")
else:
    print("a >= 100")

_result = "success" if success else "failed"
```

Assert

```
assert a != b
assert a == b, "SOS"
```

Function

```
func = lambda x: int, y: int -> int {
    x + y
}
a = func(1, 1)  # 2
```

Top-Level Argument

```
# kcl -DbankCard=123 employee.k
bankCard = option("bankCard")

# kcl main.k -D list_key='[1,2,3]' -D dict_key
      ='{"key":"value"}'
list_key = option("list_key")
dict_key = option("dict_key")
```

Arguments with Setting Files

```
# kcl -Y setting.yaml employee.k

# setting.yaml
kcl_options:
  - key: key_number
    value: 1
  - key: key_dict
    value:
      innerDictKey:  innerDictValue
  - key: key_list
    value:
      - 1
      - 2
      - 3
  - key: bankCard
    value: 123
```

Concepts DOC

KCL: Config

```
import kubernetes.core.v1

# Create a kubernetes deployment resource
deployment = v1.Deployment {
    metadata.name = "nginx"
    metadata.labels.app = metadata.name
    spec = {
        replicas = 3
        selector.matchLabels.app = metadata.name
        template = {
            metadata.labels.app = metadata.name
            spec.containers = [{
                name = metadata.name
                image = "nginx:1.14.2"
                ports = [{containerPort=80}]
            }]
        }
    }
}
```

KCL: Schema

```
import units

type UnitType = units.NumberMultiplier

# Define a schema named Resource with
# three attributes and constraints
schema Resource:
    cpu: int | UnitType = 1
    memory: UnitType = 1024Mi
    dist: UnitType = 10Gi

    check:
        0 < cpu < 64
        0 < memory <= 64 Gi
        0 < disk <= 1Ti
```

KCL: Rule

```
data: Data = option("data")
input: Input = option("input")

# Define a RBAC rule
rule Allow:
    any grant in UserIsGranted() {
        input.action == grant.action and input.
              type == grant.type
    }
    any user in data.user_roles[input.user] {
        user == "admin"
    }

rule UserIsGranted:
    [   grant
        for role in data.user_roles[input.user]
        for grant in data.role_grants[role]
    ]

allow = Allow or False
```

KCL: Lambda

```
# Transform input resource  and change annotation

transformer = lambda res {
    res | {
        metadata.annotations: {
            "managed-by" = "kcl"
        }
    } if res.kind == "Deployment" else res
}

output = [transformer(res) for res in option("
      input")]
```

KCL = Config + Schema + Rule + Lambda