# IOTA ACADEMY
**Become Job Ready...**

# INTRODUCTION: PART - 1

## Python Keywords

- Keywords are the **reserved words in Python**.
- We **can't use** a keyword as a *variable name, function name or any other identifier*.
- Keywords are **case-sensitive**.

In [16]:
```
# Get all keywords

import keyword    # keyword library

print(keyword.kwlist)

print("\nTotal number of keywords: ", len(keyword.kwlist))
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continu
e', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'la
mbda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

Total number of keywords:  36
```

In [24]:
```
# examples: (can't use keyword as variable name)

global = 10
```

```
  Cell In[24], line 3
    global = 10
           ^
SyntaxError: invalid syntax
```

In [18]:
```
# example: (case sensitive)

global = 10

print(globaL)
```

```
10
```

## Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

**Rules for writing identifiers:**

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase A to Z or digits (0 to 9) or an underscore( _ ).
2. An identifier cannot start with a digit. Example: 1var is invalid, but var1 is perfectly fine.
3. Keywords cannot be used as identifiers.
4. We cannot use special symbols like !, @, #, $, % etc. anywhere in our identifier.

In [19]:
```
# example: nameOfAcademy and x1 are variables

name = "iota"
print(name)
```

```
iota
```

## Python Comments

Comments are lines that exist in the computer programs that **are ignored by compilers and interpreters.**

### Use of Comments:

1. It makes code more readable for humans as it provides some information or explanation about what each part of a program is doing.
2. Useful in longterm (as it is to forget what we did previously).

In Python, we use **hash (#) symbol to start writing a comment**.

In [20]:
```
# print hello iota to the console

print("hello iota")
```

```
hello iota
```

### Multi-line Comments

**Way 1:**

Use hash (#) in the begining of each line.

**Way 2:**

Use triple quotes, either `'''` **or** `"""`. Make sure you add closing quotes too.

In [21]:
```
# Way1
# this is an
# example of
# multi-linecomment


'''
Way 2:
I am learning Python with IOTA Academy.
It's fun :)
'''

x = 30
print(x)
```

```
30
```

# Python Indentation

1. Most programming languages like C, C++, Java use braces { } to define a block of code. **Python uses Indentation.**
2. A code block (body of a function, loop, if-else etc.) starts with indentation and ends with the first unintended line.

Instead of referring to a code "block," Python programmers use the word "suite." Both names are used in practice, but the Python docs prefer "suite."

> *the amount of indentation is up to you, but it must be consistent throughout that block.*

1. Generally **four whitespaces** are used for indentation and is preferred over tabs.

In [22]:
```
# level 1

x = 20

if x>10:
    print(x-2)
    print("hello")

else:
    print(x)
```

```
18
hello
```

In [23]:
```
# inconsistent indentation gives indentation error

a = 10
  b = 20
print(a,b)
```

```
  Cell In[23], line 4
    b = 20
    ^
IndentationError: unexpected indent
```

*Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable.*

In [25]:
```
# with indentation

if True:
    print("iota academy")
    course = "Python"
```

```
iota academy
```

In [26]:
```
# without indentation

if True: print("iota academy"); course = "Python"
```

```
iota academy
```

## Python Statement

Instructions that a Python interpreter can execute are called statements.

Examples:

In [27]:
```
a = 1    # single line statement
```

### Multi-line statements

In Python, end of a statement is marked by a newline character (\n). But we can make a statement extend over multiple lines with the line continuation character \ or parenthesis ().

In [28]:
```
# using backslash

a = 1+2+3 \
    + 4+5+6+7 \
    + 8+9+10+11+12
print(a)
```

```
84
```

In [29]:
```
# we can use parenthesis ()

b = (1+2+3+4+5
     +6+7+8
     +9+10+11)
print(b)
```

```
66
```

In [30]:
```
# multiple statements in single line using; (semi-colon)

a = 10;  b = 20; c = 30

print(a,b,c)
```

```
10 20 30
```

In [31]:
```
# this also have same result as above

a = 10
b = 20
c = 30
print(a,b,c)
```

```
10 20 30
```

## Variables

A variable is a location in the memory used to store some data (value).

1. They are given unique names to differentiate between different memory locations.
2. The rules for writing a variable is same as the rules for writing identifiers in Python.
3. We don't need to declare a variable before using it.

> *In Python, we simply assign a value to a variable and it will exist. We don't even have to declare the type of variable. This is handled internally according to the type of value we assign to the variable.*

## Variable Assignments

*Syntax: variable_name = value or data here*

We use *assignment* operator (=) to assign values to a variable

In [32]:
```
# Snake Case: Each word is separated by an underscore character
variable_name = "I am a variable of snake case"

# Camel Case: Each word, except the first, starts with a capital letter
variableName = "I am a variable of camel case"

# Pascal Case: Each word starts with a capital letter We use pascal case in class name
VariableName = "I am variable of pascal case"
```

In [33]:
```
# Example
a = 10      # int
b = 5.5     # float
c = "iota"  # string (str)

# use type function to check datatype of variable
print(type(a))
print(type(b))
print(type(c))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

### Multiple Assignments

In [34]:
```
a, b, c = 10, 5.5, "iota"    # same operation as previous
```

In [35]:
```
a = b = c = 10               # same value to all variables at once.
```

## Storage Locations

Can be accessed using **id** function.

In [36]:
```
x = 3
print(id(x))    # prints address of x in the memory location
```

```
1991400515952
```

In [37]:
```
y = 3
print(id(y))    # prints address of y in the memory location
```

```
1991400515952
```

In [38]:
```
x = x+3         # = 3 + 3 =6
print(x)
print(id(x))
```

```
6
1991400516048
```

In [39]:
```
print(y)
print(id(y))
```

```
3
1991400515952
```

**Observation:**

> x & y points to the same location

*Python internally stores value and not the variable; variables are pointers of that location.*

In [40]:
```
# Example 2 (notice effect of sharing same memory location)

x = [1,2,3,5,6]

y = x

print(x)
print(id(x))
print(y)
print(id(y))    # both have same id
```

```
[1, 2, 3, 5, 6]
1991505953024
[1, 2, 3, 5, 6]
1991505953024
```

In [41]:
```
y.append("iota")

print(y)
print(x)          # notice the change in x
```

```
[1, 2, 3, 5, 6, 'iota']
[1, 2, 3, 5, 6, 'iota']
```

In [42]:
```
y = y + [4]      # assigning new value to the same variable using = operator

print(x)
print(y)
print(id(x))
print(id(y))
```

```
[1, 2, 3, 5, 6, 'iota']
[1, 2, 3, 5, 6, 'iota', 4]
1991505953024
1991505953216
```

# Exercise: 1

1) Create a variable named *iota* and assign the value "Data Analytics Institute" to it.

2) Create three variables *x, y* and *z* assign same value 44 to all the variables.

3) Create three variables *p, q,* and *r* and assign 45 to p, 56 to q and 87 to r.

4) Create a variable *iota* and assign a value 100 to it and prints its id.

5) Create a variable *'iota'* and assign it value "for Python". In the next line create the same variable *'iota'* and assign it "for Data Analytics". Now print the variable *'iota'* and check its output. Discuss the output with your batch-mates.

# Great Job!