Variables

Variables are containers for storing data values.

Creating variables in Python is very simple: we type the name of the variable, then an equals sign, then put the value that we are assigning to that variable on the right of equals sign.

A variable in Python can hold any type of data, this could be a primitive value (a number or character), or an object (a string, sprite, list, etc).

In [1]:
```
score = 0
name = "Mr. Powers"
aList = [1, 2, 3, 4, 5]
aTuple = (1, 2, 3, 4, 5)
```

Rules of Variables

Variable names can contain only letters, numbers, and underscores.

Note: They can start with a letter or an underscore, but not with a number.

In [2]:
```
message_1 = "correct"
```

In [3]:
```
1_message = "incorrect"
```

```
  Cell In[3], line 1
    1_message = "incorrect"
     ^
SyntaxError: invalid decimal literal
```

In [4]:
```
message2@="incorrect"
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 message2@="incorrect"

NameError: name 'message2' is not defined
```

Spaces are not allowed in variable names, but underscores can be used to separate words in variable names.

In [5]:
```
greeting_message = "correct"
greeting message = "incorrect"
```

```
  Cell In[5], line 2
    greeting message = "incorrect"
                ^
SyntaxError: invalid syntax
```

Avoid using Python keywords and function names as variable names;
Do not use words that Python has reserved for a particular programmatic purpose

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

In [11]:
```
#print="Correct but dont use"
print("blabla")
aand = "incorrect"
print(aand + 'Hello')
print(aand)
print(aand)
```

```
blabla
incorrectHello
incorrect
incorrect
```

Variable names should be short but descriptive. For example, name is better than n, student_name is better than s_n, and name_length is better than length_of_persons_name.

In [12]:
```
n = "bad"
```

```
name = "good"
s = "bad"
s_n = "bad"
student_name = "good"
length_of_persons_name = "ok"
name_length = 'better'
```

Be careful when using the lowercase letter l and the uppercase letter O because they could be confused with the numbers 1 and 0.

In [13]:
```
message = "this will not work"
print(mesage)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 2
      1 message = "this will not work"
----> 2 print(mesage)

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

In [14]:
```
mesage = 'HELLO ETE 4990!'
print(mesage)
print("You can also do this", mesage)
print(mesage.title())
print(mesage.find("4"))
```

```
HELLO ETE 4990!
You can also do this HELLO ETE 4990!
Hello Ete 4990!
10
```

Methods

A method is an action that Python can perform on a piece of data.

The dot after the mesage tells Python to perform the title() method on the variable.

It is important to note that a method is allwas followed by a set of parentheses. In many cases the method needs additional information to perform an action.

In this case the parentheses are empty but wat about the method print()

Print Statments

In [15]:
```
print("Score: ", score, type(score)," Name: ", name, type(name),
      "  A List:", aList, type(aList), "  A Tuple:", aTuple, type(aTuple))
print("\tScore:\t", score, type(score))
print("\tName:\t", name, type(name))
print("\tA List:\t", aList, type(aList))
print("\tA Tuple:", aTuple, type(aTuple))
```

```
Score:  0 <class 'int'>   Name:  good <class 'str'>   A List: [1, 2, 3, 4,
5] <class 'list'>   A Tuple: (1, 2, 3, 4, 5) <class 'tuple'>
        Score:   0 <class 'int'>
        Name:    good <class 'str'>
        A List:  [1, 2, 3, 4, 5] <class 'list'>
        A Tuple: (1, 2, 3, 4, 5) <class 'tuple'>
```

In [20]:
```
print(name,str(score))
#help(print)
print(name,str(score), sep="")
```

```
good 0
good0
```

Global and Local variables

In Python it is fairly simple to create and assign values to variables, however it is a little more complex to use them. Here we will discuss what is known as scope .

Scope is the area of code in a program where a certain (variable) name is valid, meaning itcan be used.

Mainly there are 2 types of scopes for variables Global and Local

In [29]:
```
def change_score():
    score = 10
    #global score
    score+=1
    print(score)
```

In [ ]:
```

```

In [43]:
```
print(score)
change_score()
print(score)
```

```
1
101
1
```

In [47]:
```
wakeup()
```

```
wake up at 10
```

In [49]:
```
def wakeup(alarm_time=None):
    """ This code sets your alarm if no alarm
    set then set alarm to 10 by default"""
    if alarm_time:
        alarm = alarm_time
    else:
        alarm = 10
    #global score
    print("wake up at", alarm)
```

In [50]:
```
help(wakeup)
```

```
Help on function wakeup in module __main__:

wakeup(alarm_time=None)
    This code sets your alarm if no alarm
    set then set alarm to 10 by default
```

if statments

An If statment is the heart of every function. It is a conditional test and it uses the values of True and False to deside to exicute the code.

In [ ]:
```
name == "Mr. Powers"
```

In [ ]:
```
name == "MR. Powers"
```

In [ ]:
```
name.lower() == "mr. powers"
```

In [ ]:
```
print(name)
```

An if statment has the following setup

```
if conditional_test:
       do something
```

In [ ]:
```
if name != "Bill Nie":
    print("you are not a famous TV star")
```

In [ ]:
```
if score !=0:
    print("score is not 0")
```

```
else:
    print("score is 0")
```

In [ ]:
```
score+=1
print(score>=0)
print(score<0)
print(score>0)
print("Hello" =="olleH")
```

In [52]:
```
print(score>=0 and aList[0]==1)
print(True or False)
print(False and False)
```

```
True
True
False
```

In [55]:
```
aList
```

Out[55]:
```
[1, 2, 3, 4, 5]
```

In [53]:
```
2 in aList
```

Out[53]:
```
True
```

In [54]:
```
6 in aList
```

Out[54]:
```
False
```

In [ ]:
```
if 6 not in aTuple:
    print("six is not in aList")
else:
    print("six is in aList")
```

In [ ]:
```
aTuple=(6,3)
```

In [ ]:
```

```