Python overview

Hello world

- First step is to make
- Makes sure your environment is working
- If you get are a response, it means tool chain is functioning

Python Anatomy

- Simple scripting language
- Common layout for a script
  - **Comments** to explain the code (starts with #)
  - Import **modules**
  - Then the script as combination of **statements** and **functions**

Expression and Statements

- Generally speaking statement: unit of execution. and expression is unit of evaluation
- In Python:
  - **Expression** is any combination of literals, identifiers, and operators. Anything that returns a value

a = b	(assignment expression)	*(a, b)*	(aggregate value expression)
a + b	(operator expression)	*a*	(simple value expression)
True	(built-in constant val. expression)	*f ()*	(function expression)

  - **Statement** is a line of code that may be an expression or not (eg. import, break, continue statements are not expressions

Whitespace and Comments

- Lateral white space is significant
  - A block is delimited by indentation (instead of parenthesis or curly brackets)
  - Indentation signify a function, conditional statements or loops
- Horizontal white space is insignificant

In [48]:

```
1  for a_letter in 'yegin':
2      print(a_letter)      ## inside the loop
3  print('All DONE!')       ## outside of the loop
4
```

```
y
e
g
i
n
All DONE!
```

In [ ]:

```
1
```

Blocks and Scope

- More on indentation
- Executions of a statement (a line) depends whether the block it is in executed
- Part-way indentations will cause errors

In [53]:

```
1 x=10
2 y=11
```

In [57]:

```
1 if x> y :
2     print('x>y')
3     print('x is greater ')
4 else:
5     print('y>x')
6     print('y is greater')
```

```
y>x
y is greater
```

In [59]:

```
1 if x> y :
2     print('x>y')
3   print('x is greater ')
4 else:
5     print('y>x')
6     print('y is greater')
```

```
  File "<ipython-input-59-1ecb5bdc1171>",
line 3
    print('x is greater ')
                         ^
IndentationError: unindent does not match
any outer indentation level
```

Conditionals

- Dictates whatever statement or block comes next is executed
- Use ”:” to end
- No *switch* or case *statement*, it has “elif”

In [60]:

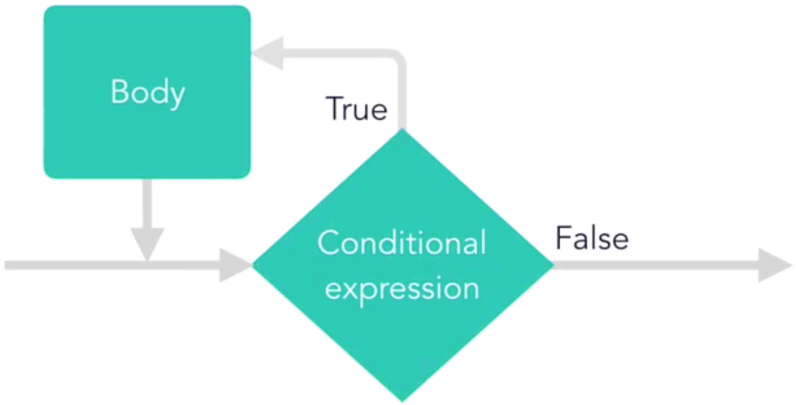
```
1  if x> y :  ## column indicates end of condition
2      print('x>y')
3      print('x is greater ')
4  elif x== y: ## will check only if the previous
5                ## condition is false
6      print('x equals to y ')
7  else:
8      print('y>x')
9      print('y is greater')
```

```
y>x
y is greater
```

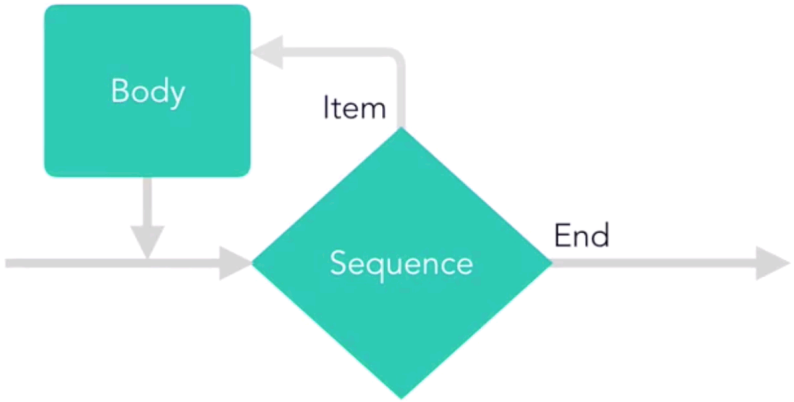
In [ ]:

```
1
```

Loops



while loop



for loop

Functions

- Like functions and subroutines in other languages
- We will use them for reusability
- A function typically has inputs (variables), and returns an output, but it doesn’t have to.

```
1  def print_name():     ## no input
2      print('yegin ')   ## doesn't return an output
3                        ## just prints
```

```
1  print_name()
```

```
yegin
```

```
1  def get_fullname(first_name, last_name):  ## inputs first name
2                                            ## and last name
3      full_name= first_name + " "+ last_name
4      return full_name                      ## returns full name
```

```
1  get_fullname('yegin', 'genc')
```

```
'yegin genc'
```

```
1  myfullname=get_fullname('yegin', 'genc')
2  print(myfullname)
```

```
yegin genc
```

Types and Values

String Type

- String is wrapped in single quote or double quote marks.
- Can apply text functions such as:
  - .upper()
  - .lower()
  - ….
  - .format()

```
1 name='yegin'
2 print(name)
```

```
yegin
```

```
1 first_name='yegin'.upper()
2 print(first_name)
```

```
YEGIN
```

```
1 last_name='genc'.title()
2 print(last_name)
```

```
Genc
```

```
1 'My name is {} {}'.format(first_name, last_name)
```

```
'My name is YEGIN Genc'
```

```
1 'My name is '+ first_name + ' '+ last_name
```

```
'My name is YEGIN Genc'
```

Numeric Types

- Two basic types are **integers** and **floats**
- Integer whole numbers
- floats have decimals
- It can be transferred from one another

```
1  y=2.0
2  print(y, type(y))
3  x=10
4  print( x,    type(x))
5  print( x+1,  type(x+1))
6  print( x/2, type(x/2))
7
8  print(x//3, type(x//3))
9  print(x%3, type(x%3))
```

```
2.0 <class 'float'>
10 <class 'int'>
11 <class 'int'>
5.0 <class 'float'>
3 <class 'int'>
1 <class 'int'>
```

Boolean Type

- Used for logical expressions
- **True** or **False**
- Comparisons return a Boolean.
  - 9 > 7 (True)
  - 5 > 10 (False)
- Special attention to **None** which is a special type of its own

```
1  x= False
2  print(x)
```

```
False
```

```
1  y=7>5
2  print(y, type(y))
```

```
True <class 'bool'>
```

```
1  if x:
2      print('statement is true')
3  else:
4      print('statement is false')
```

```
statement is false
```

Sequence Type

- Lists
  - []
  - mutable (can update its elements)
- Tuples
  - ()
  - Immutable
- Dictionaries
  - {key: value}
  - List of key value pairs

List

```
1 a=[1,2,7, 10 ]   # list
2 print(a)
3 print(a[0] , a[2])
```

```
[1, 2, 7, 10]
1 7
```

```
1 a[0]=-1
2 print(a)
```

Tuple

```
1 t= (10,20 ,30)
2 print(t[0])
```

```
10
```

```
1 t[0]=5
```

```
---------------------------------------------------------------
TypeError                                 Traceback (most re
<ipython-input-159-1c25f0002d07> in <module>()
----> 1 t[0]=5

TypeError: 'tuple' object does not support item assignment
```

```
1 prices={'orange':2,'apple':1.5, 'banana':.5 }
2 prices['orange']
```

```
2
```

Conditions

Conditional Syntax

Conditional Operators

Conditional Operators

==	a == b	Equal
!=	a != b	Not equal
<	a < b	Less than
>	a > b	Greater than
<=	a <= b	Less than or equal
>=	a >= b	Greater than or equal

Logical Operators

and	x and y	True if both x and y
or	x or y	True if x or y
not	not x	Invert state

Membership Operators

x in y

True if x member of collection y

x not in y

True if x not member of collection y

Operators

Arithmetic operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer Division
%	Remainder (Modulo)
**	Exponent
-	Unary negative
+	Unary positive

Comparison operators

<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
==	Equal
!=	Not equal

Boolean operators

`and`	And
`or`	Or
`not`	Not
`in`	Value in set
`not in`	Value not in set
`is`	Same object identity
`is not`	Not same object identity

Operator precedence

10 + 3 * 2 = ?

If in doubt use parenthesis…