Let’s Dive In !!

Installing Python

Windows:

- Download Python from http://www.python.org
- Install Python.
- Run Idle from the Start Menu.

Linux:

- Chances are you already have Python installed. To check, run python from the terminal.
- If not, install from your distribution's package system.

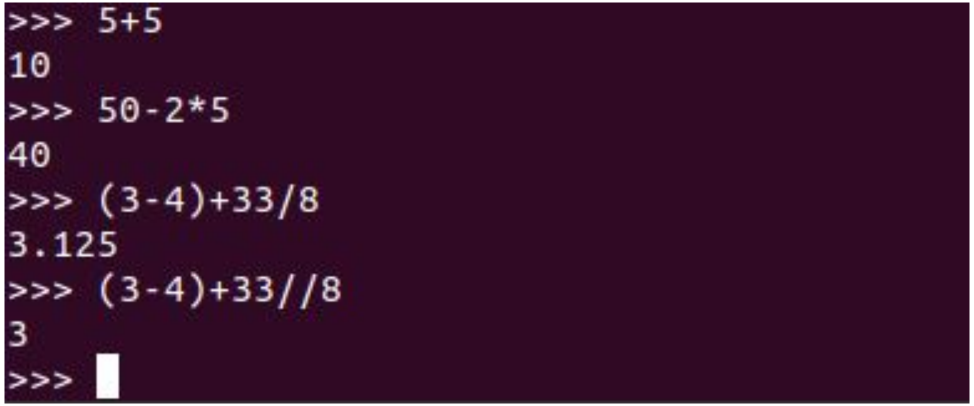
Mac OS X:

- Python is already installed.
- Open a terminal and run python or run Idle from Finder.

Installation Instructions

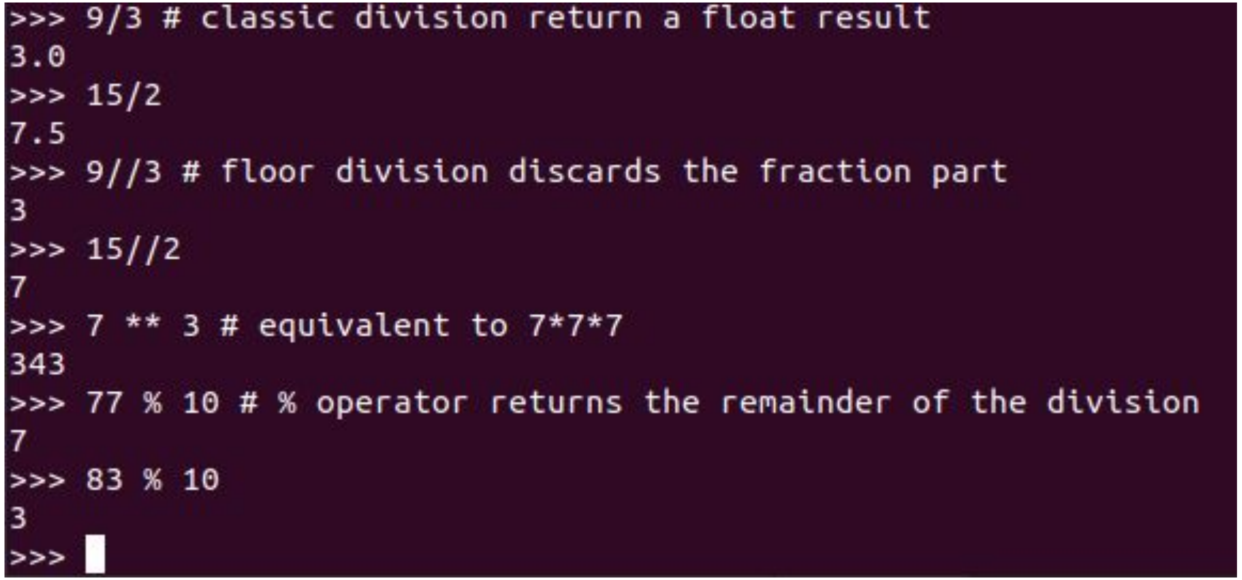
Using Python as a Calculator

- Can act as a simple calculator
- type an expression at it and it will write the value



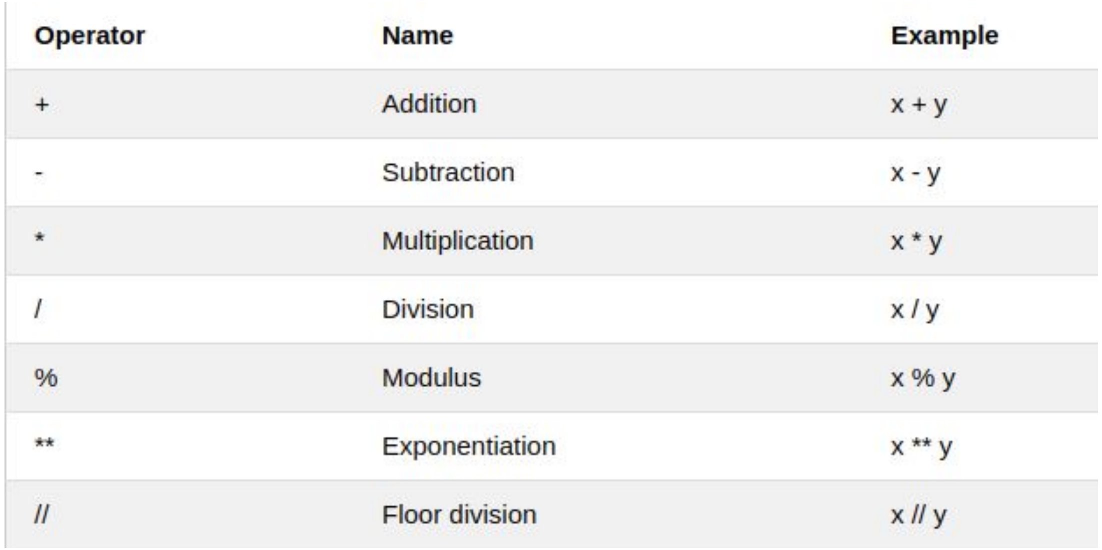
```
>>> 5+5
10
>>> 50-2*5
40
>>> (3-4)+33/8
3.125
>>> (3-4)+33//8
3
>>> 
```

Some more mathematical calculations



```
>>> 9/3 # classic division return a float result
3.0
>>> 15/2
7.5
>>> 9//3 # floor division discards the fraction part
3
>>> 15//2
7
>>> 7 ** 3 # equivalent to 7*7*7
343
>>> 77 % 10 # % operator returns the remainder of the division
7
>>> 83 % 10
3
>>> 
```

Python Operators



Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

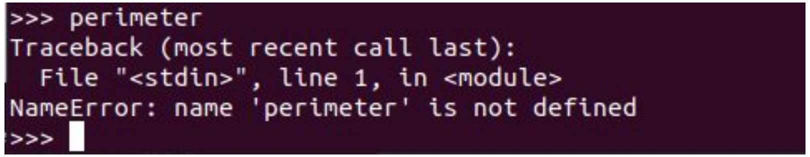
Variables

- A variable is a named memory location
- Used to store values
- Analogy: think variable as a box, values of variable can be thought as the content of the box

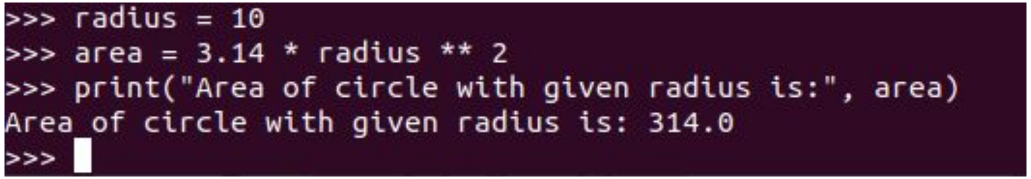
Assignment Statement

- They are used to assign values to variables.
- The ‘=’ symbol indicates assignment
- The assignment statement r = 10 creates the variable r and assigns to it the value of 10.

- What happen if we do not assign value to variable?



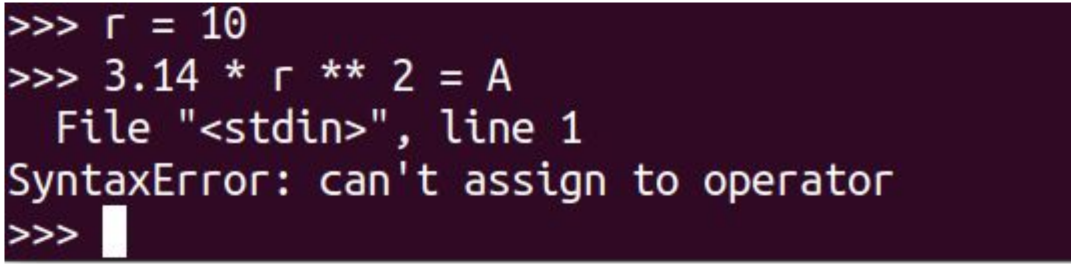
```
>>> perimeter
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'perimeter' is not defined
>>> 
```



```
>>> radius = 10
>>> area = 3.14 * radius ** 2
>>> print("Area of circle with given radius is:", area)
Area of circle with given radius is: 314.0
>>> 
```

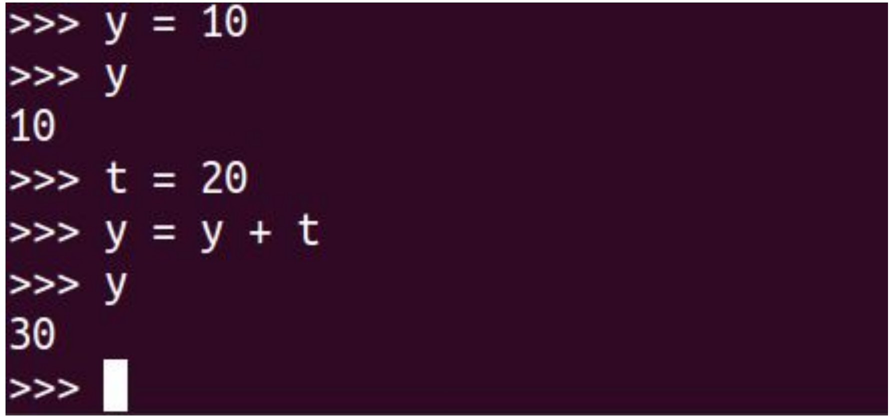
Assignment vs “Is Equal to”

- In Math “=“ is used to say what is on the left equals what is on the right.
- In Python, “=“ prescribes an action, “evaluate the expression on the right and assign its value to the variable named on the left.”



```
>>> r = 10
>>> 3.14 * r ** 2 = A
  File "<stdin>", line 1
SyntaxError: can't assign to operator
>>> 
```

Updating the variables



```
>>> y = 10
>>> y
10
>>> t = 20
>>> y = y + t
>>> y
30
>>> 
```

Assignment vs Equations

- In algebra,

  t = t +10 doesn’t make sense unless you believe 0 =t-t = 10

- In Python,

  t = t + 10 means add 10 to the value of t and store the result in t.

2 Step Action Behind Every Assignment Statement

< *variable name* > = < *expression* >

- Evaluate the expression on the right hand side.
- Store the result in the variable named on the left hand side.

Precedence

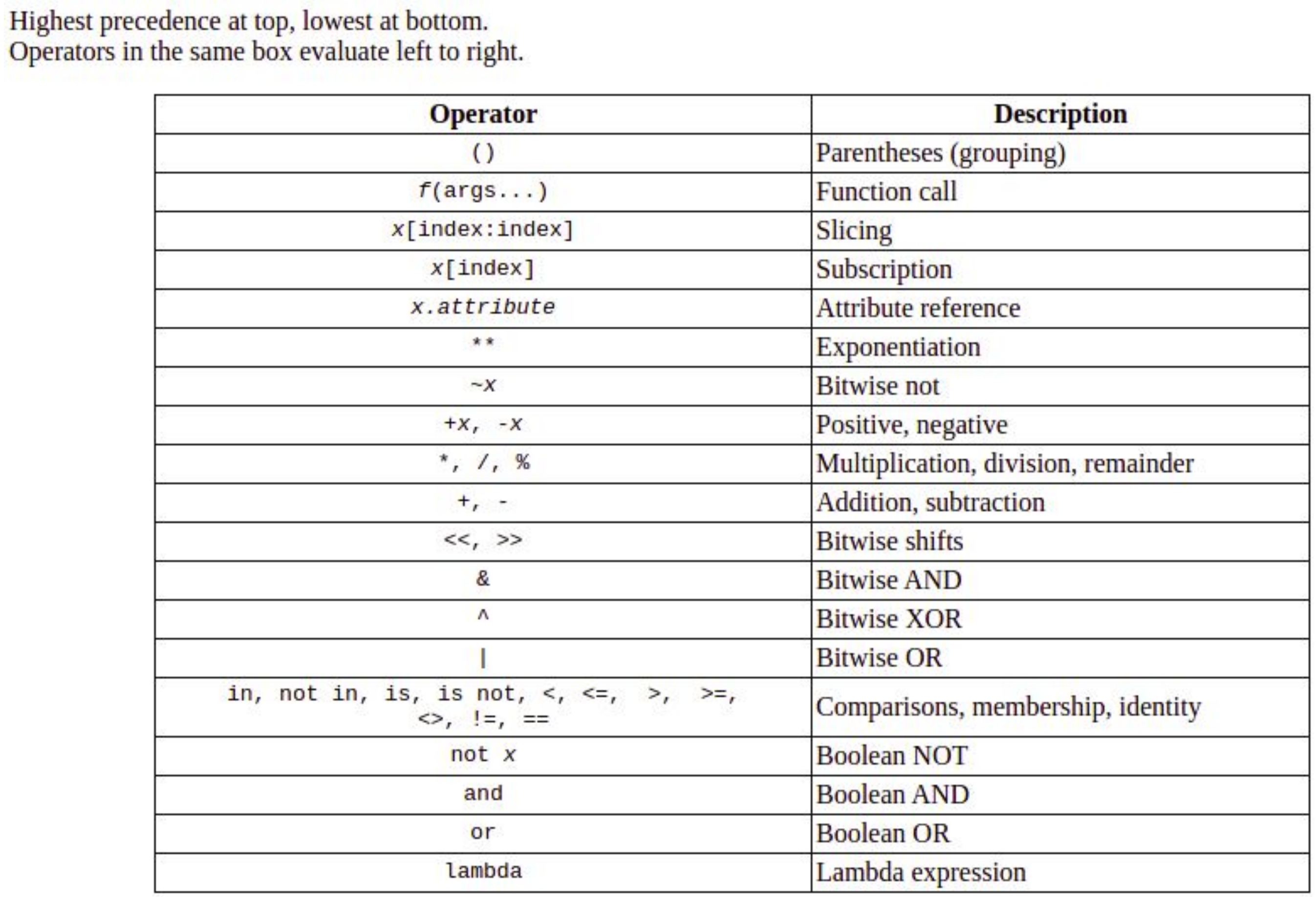
What is the order of execution of an expression?

This:

- A + B*C
- A**2/4
- A*B/C*D

Is the same as:

- A + (B*C)
- (A**2)/4
- ((A*B)/C)*D



Highest precedence at top, lowest at bottom.
Operators in the same box evaluate left to right.

Operator	Description
`()`	Parentheses (grouping)
`f(args...)`	Function call
`x[index:index]`	Slicing
`x[index]`	Subscription
`x.attribute`	Attribute reference
`**`	Exponentiation
`~x`	Bitwise not
`+x, -x`	Positive, negative
`*, /, %`	Multiplication, division, remainder
`+, -`	Addition, subtraction
`<<, >>`	Bitwise shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`in, not in, is, is not, <, <=, >, >=, <>, !=, ==`	Comparisons, membership, identity
`not x`	Boolean NOT
`and`	Boolean AND
`or`	Boolean OR
`lambda`	Lambda expression

source

Associativity

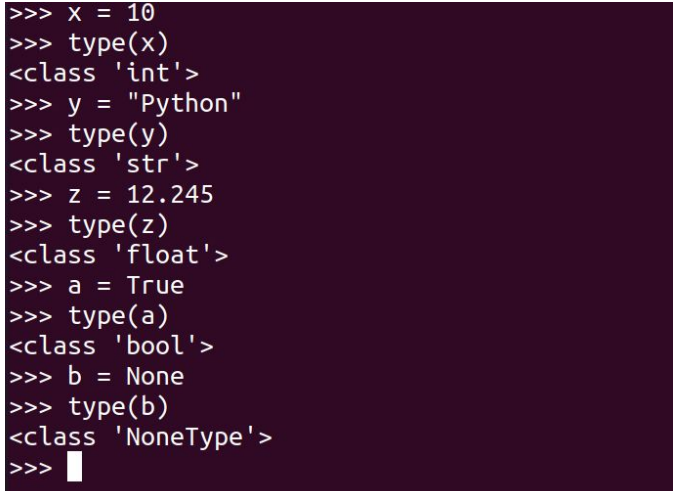
(order of execution of operators)

- Almost all the operators have left-to-right associativity
- `**` has right-to-left associativity

- When two operators share an operand and the operators have the same precedence, then the expression is evaluated according to the *associativity* of the operators.
- For example, since the `**` operator has right-to-left associativity, `a ** b ** c` is treated as `a ** (b ** c)`
- On the other hand, since the `/` operator has left-to-right associativity, `a / b / c` is treated as `(a / b) / c`

Data Types

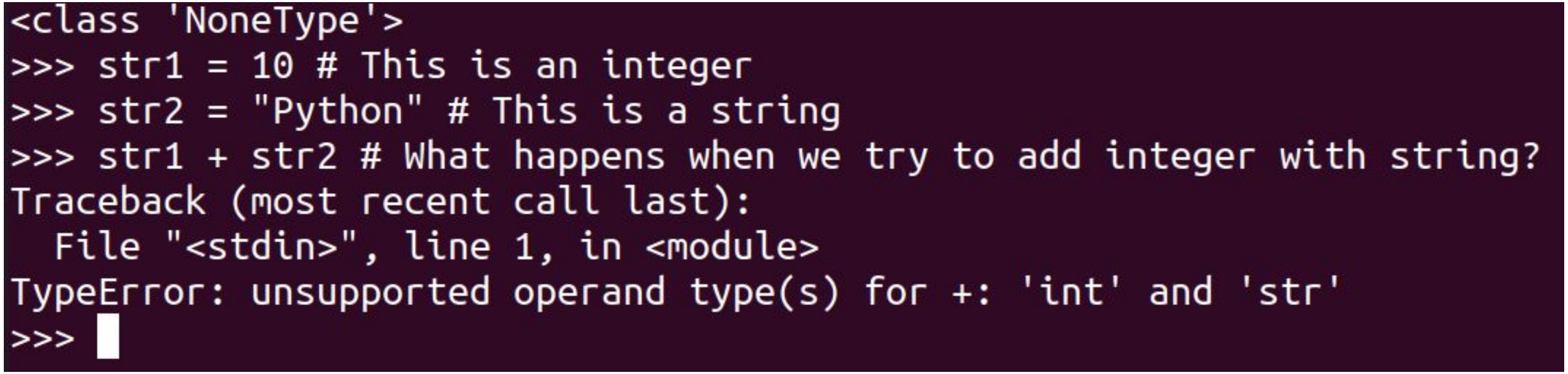
- Variables has a type, which is defined the way it store values.
- If 10 is assigned to a variable ‘x’, then the type of x is ‘int’.
- Similarly, if ‘python’ is assigned to variable ‘y’, type of y becomes ‘str’



```
>>> x = 10
>>> type(x)
<class 'int'>
>>> y = "Python"
>>> type(y)
<class 'str'>
>>> z = 12.245
>>> type(z)
<class 'float'>
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = None
>>> type(b)
<class 'NoneType'>
>>> 
```

Why should we care about data types?

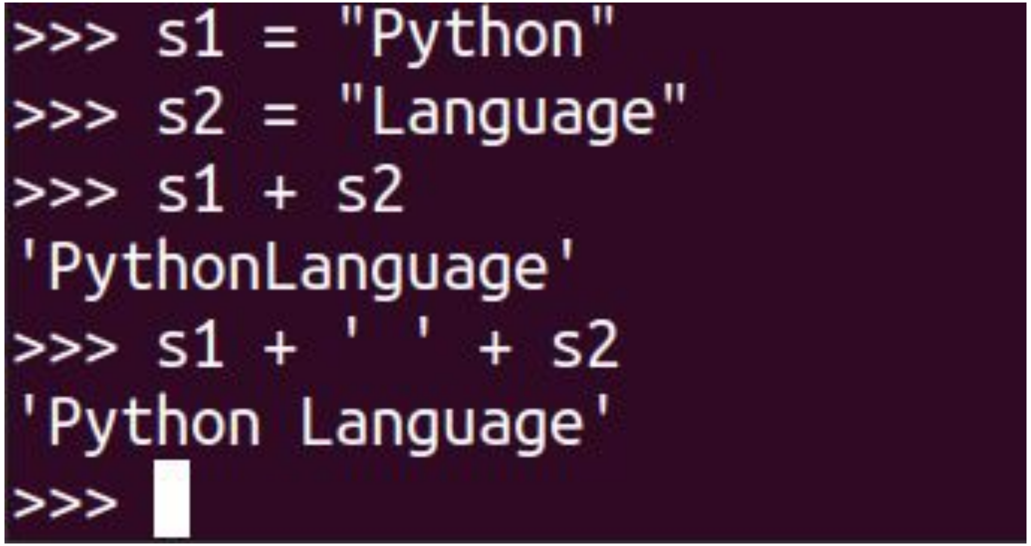
- We simply cannot do arithmetic operations between variables of different types. It leads to an error.



```
<class 'NoneType'>
>>> str1 = 10 # This is an integer
>>> str2 = "Python" # This is a string
>>> str1 + str2 # What happens when we try to add integer with string?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

Strings

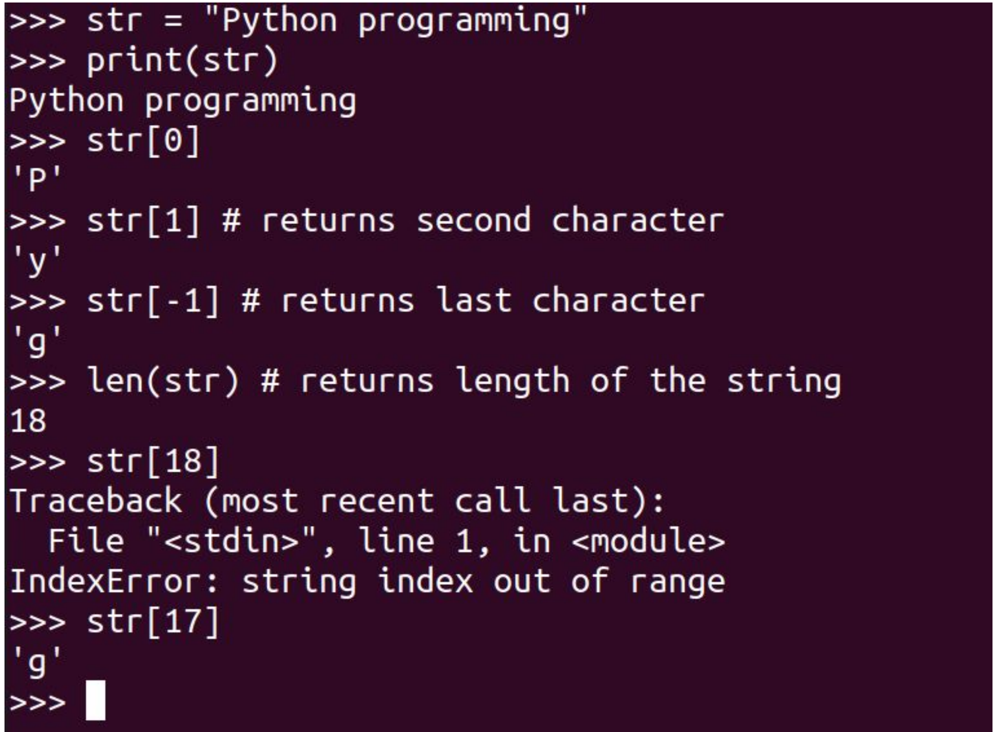
- Used to represent text
- They are quoted characters
- May be single quoted (‘   ‘) or double quoted (“   “)



```
>>> s1 = "Python"
>>> s2 = "Language"
>>> s1 + s2
'PythonLanguage'
>>> s1 + ' ' + s2
'Python Language'
>>> 
```

Indexing

- In python, indexing starts with 0 and go through n-1 where n is the length of the string.

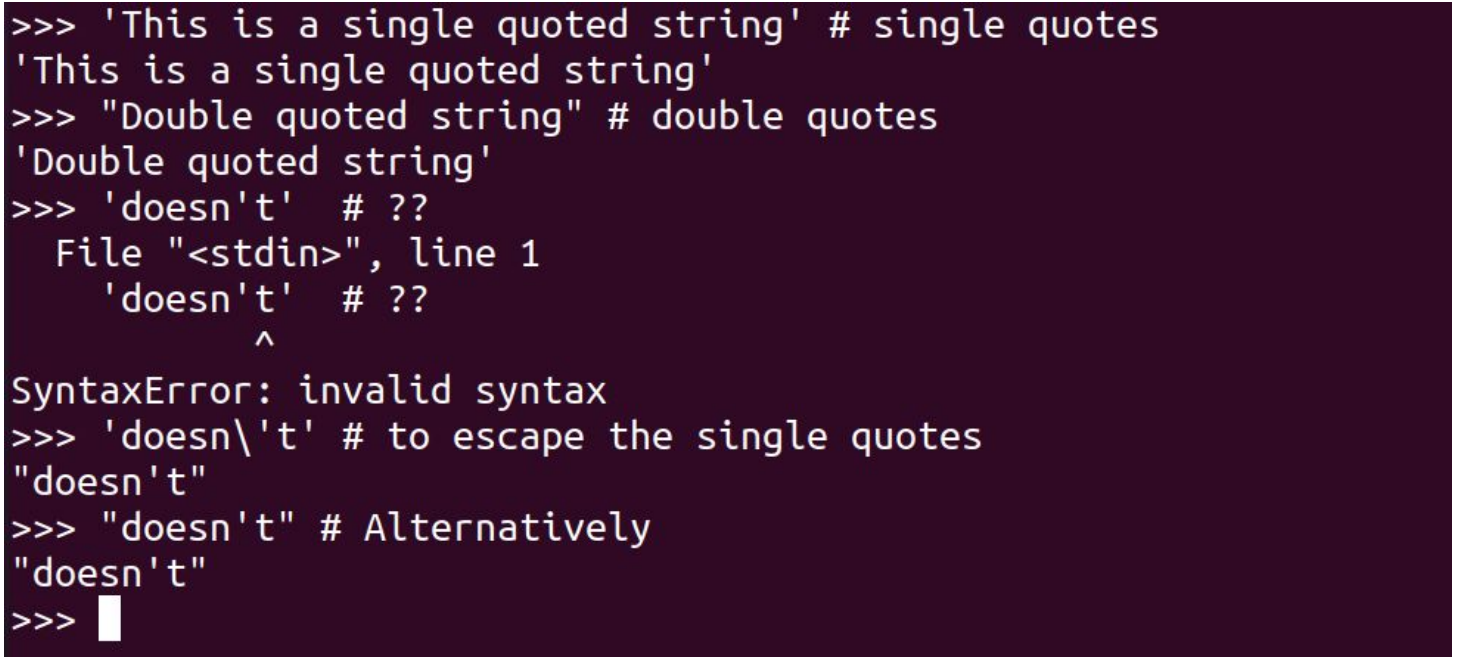


```
>>> str = "Python programming"
>>> print(str)
Python programming
>>> str[0]
'P'
>>> str[1] # returns second character
'y'
>>> str[-1] # returns last character
'g'
>>> len(str) # returns length of the string
18
>>> str[18]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> str[17]
'g'
>>> 
```

String Operations

- String can be added (called as concatenation)
- String can also be multiplied. It creates a copy of the same string multiple times
- It can be compared using relational operators
- Check if substrings are present in given string using keyword ‘in’
- Long strings that span multiple lines can be made using '''

Some Code for strings !!



```
>>> 'This is a single quoted string' # single quotes
'This is a single quoted string'
>>> "Double quoted string" # double quotes
'Double quoted string'
>>> 'doesn't'  # ??
  File "<stdin>", line 1
    'doesn't'  # ??
          ^
SyntaxError: invalid syntax
>>> 'doesn\'t' # to escape the single quotes
"doesn't"
>>> "doesn't" # Alternatively
"doesn't"
>>> 
```

String operations code in notebook!!