



# Summer Coding School 2023

Aug 5, 2023

**Taehee Jeong, Ph.D.**

# “Hello, World!”

# Task:

Print "Hello, World!"

# In Jupyter notebook

```
print('Hello, World!')
```

# Using a text editor, make a file “hello.py”

In the terminal, `python hello.py`

# Python Challenge

- Constant
- Variable assignment
- Arithmetic operators
- Quiz

# Constants

- Fixed values such as numbers, letters, and strings, are called *constants* because their value does not change.
- Numeric constants are as you expect
- String constants use single quotes (') or double quotes (")

```
>>> print(123)
123
>>> print(98.6)
98.6
>>> print('Hello world')
Hello world
```

# Reserved Words

- You cannot use reserved words as variable names / identifiers.

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

# Variables

- A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”
- Programmers get to choose the names of the variables
- You can change the contents of a variable in a later statement

```
x = 12.2  
y = 14  
x = 100
```

# Python Variable Name Rules

- Must start with a letter or underscore \_
- Must consist of letters, numbers, and underscores
- Case Sensitive

Good:	spam	eggs	spam23	_speed
Bad:	23spam	#sign	var.12	
Different:	spam	Spam	SPAM	

# Sentences or Lines

```
x = 2
```

*Assignment statement*

```
x = x + 2
```

*Assignment with expression*

```
print(x)
```

*Print statement*

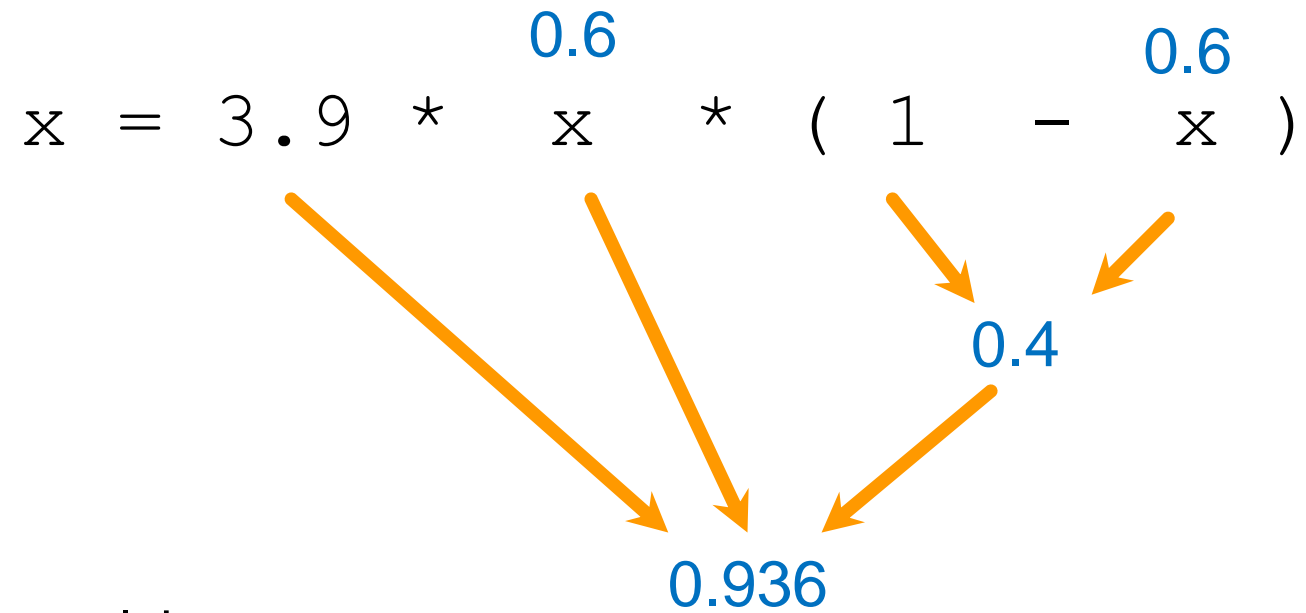


# Assignment Statements

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right-hand side and a variable to store the result

```
x = 3.9 * x * ( 1 - x )
```

A variable is a memory location used to store a value (0.6)



The right side is an expression.  
Once the expression is evaluated, the result is placed in (assigned to)  $x$ .

A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.936).

$$x = 3.9 * x * (1 - \frac{0.6}{x})$$

The diagram illustrates the evaluation of the expression  $x = 3.9 * x * (1 - \frac{0.6}{x})$ . It shows the intermediate steps of the calculation. The value 0.6 is used to calculate  $1 - \frac{0.6}{x}$ , which results in 0.4. Then, 3.9 is multiplied by the current value of x (0.6), and the result is multiplied by 0.4 to produce the final value 0.936.

The right side is an expression. Once the expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e., x).

# Arithmetic Operators

- Because of the lack of mathematical symbols on computer keyboards - we use **computer-speak** to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different than in math

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Arithmetic Operators

```
>>> xx = 2
>>> xx = xx + 2
>>> print(xx)
4

>>> yy = 440 * 12
>>> print(yy)
5280

>>> zz = yy / 1000
>>> print(zz)
5.28
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print(kk)
3

>>> print(4 ** 3)
64
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Order of operators

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others?

$x = 1 + 2 * 3 - 4 / 5 ** 6$
------------------------------

# Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print(x)
11.0
```

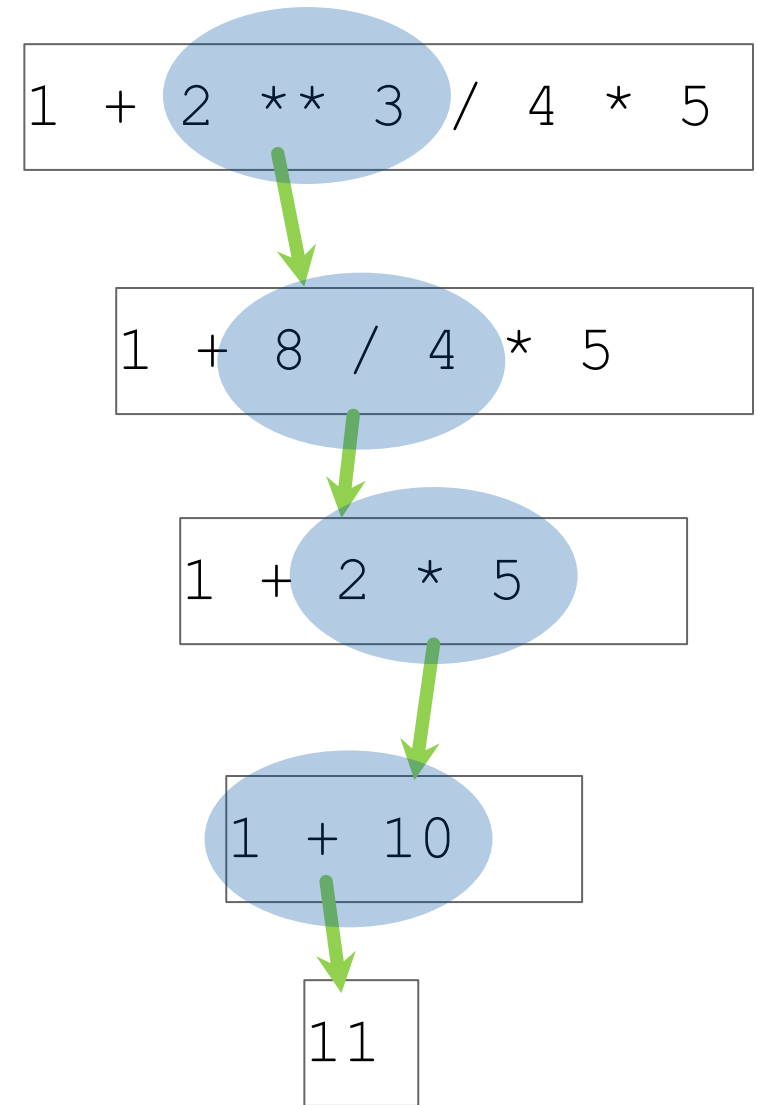
*Parenthesis*

*Power*

*Multiplication*

*Addition*

*Left to Right*





# Comments in Python

- Anything after `#` is ignored by Python
- Why comment?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code - perhaps temporarily

# Break

# What Does “Type” Mean?

- In Python, variables and constants have a **type**
- Python knows the difference between an integer number and a string
- For example “+” means:
  - **addition** if something is a number
  - **concatenate** if something is a string

```
>>> ddd = 1 + 4
>>> print(ddd)
5
>>> eee = 'hello ' + 'there'
>>> print(eee)
hello there
```

# Type Matters

- Python knows what “type” everything is
- Some operations are prohibited
- You cannot “add 1” to a string
- We can ask Python what type something is by using the `type()` function

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):  File
"<stdin>", line 1, in <module>TypeError:
Can't convert 'int' object to str
implicitly
>>> type(eee)
<class'str'>
>>> type('hello')
<class'str'>
>>> type(1)
<class'int'>
>>>
```

# Several Types of Numbers

- Numbers have two main types
  - Integers are whole numbers:  
-14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts:  
-2.5 , 0.0, 98.6, 14.0
- There are other number types:
  - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<class 'int'>
>>> temp = 98.6
>>> type(temp)
<class'float'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class'float'>
```

# Type Conversions

- When you put an integer and floating point in an expression, the integer is implicitly converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
```

# Integer Division

- Integer division produces a floating point result

```
>>> print(10 / 2)
5.0
>>> print(9 / 2)
4.5
>>> print(99 / 100)
0.99
>>> print(10.0 / 2.0)
5.0
>>> print(99.0 / 100.0)
0.99
```

# Booleans

- Logical value that indicates **True** or **False**
- Important for control flow & logic

## Logical Operators (and, or, not)

```
In [15]: 1 True and True
```

```
Out[15]: True
```

```
In [16]: 1 True and False
```

```
Out[16]: False
```

```
In [18]: 1 True or False
```

```
Out[18]: True
```

```
In [19]: 1 False or False
```

```
Out[19]: False
```

```
In [20]: 1 not True
```

```
Out[20]: False
```

```
In [21]: 1 not False
```

```
Out[21]: True
```



# String Conversions

- You can also use `int()` and `float()` to convert between strings and integer/float

```
>>> int('3')
>>> 3
>>> float('3')
>>> 3.
>>> str(3)
>>> '3'
```

# String Conversions

- You can also use `int()` and `float()` to convert between strings and integer/float
- You will get an error if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):  File
"<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str
implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):  File
"<stdin>", line 1, in <module>
ValueError: invalid literal for int() with
base 10: 'x'
```

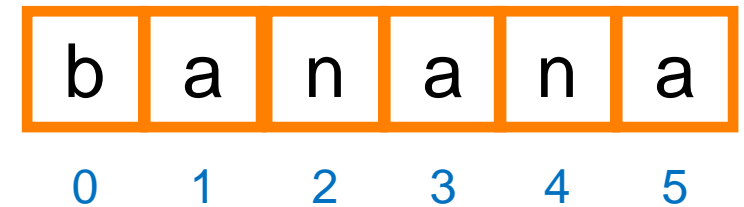
# String Data Type

- A string is a sequence of characters
- A string literal uses quotes  
'Hello' or "Hello"
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers into a string using str()

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str'
and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>> x = '123' + str(1)
>>> print(x)
```

# Looking Inside Strings

- We can get at any single character in a string using an index specified in square brackets
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed



```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

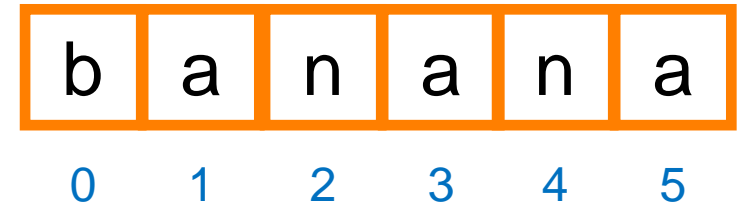
# A Character Too Far

- You will get a python error if you attempt to index beyond the end of a string
- So be careful when constructing index values and slices

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

# Strings Have Length

- The built-in function `len` gives us the length of a string

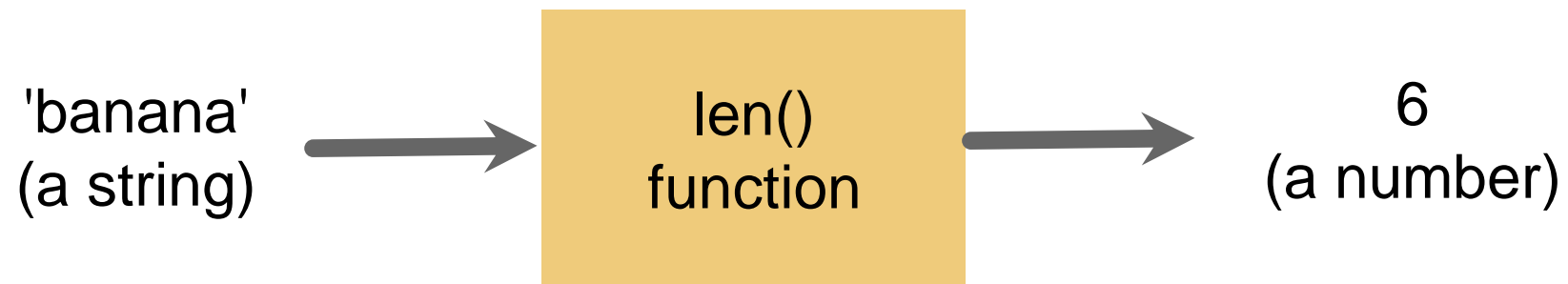


```
>>> fruit = 'banana'
>>> print(len(fruit))
6
```

# len Function

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

*A function is some stored code that we use. A function takes some input and produces an output.*



# Slicing Strings

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- We can also look at any continuous section of a string using a colon operator
- The second number is one beyond the end of the slice - “up to but not including”
- If the second number is beyond the end of the string, it stops at the end

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```



# Slicing Strings

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

# String Concatenation

When the `+` operator is applied to strings, it means “concatenation”

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
```

# Using `in` as a Logical Operator

The `in` keyword can also be used to check to see if one string is “in” another string

The `in` expression is a logical expression that returns `True` or `False` and can be used in an if statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
```

# Python Practice2\_data type

- Integer
- Floating point
- Strings
- Booleans

# Python Challenge2

- Reverse Words in a Sentence
- "Hello World" → "World Hello"

# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Modification: Taehee Jeong, San Jose State University

