Python Basics

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

Python!

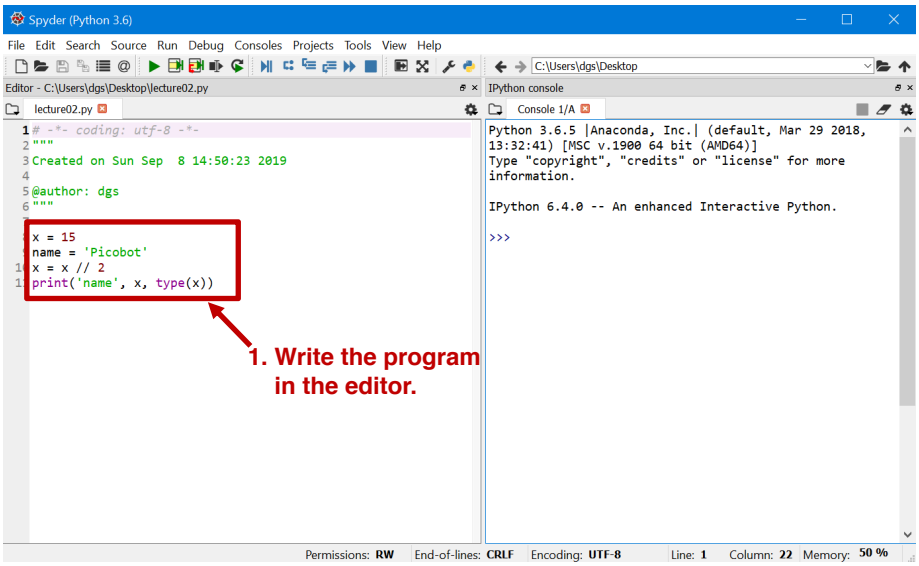
One possibility...

Happy co-existence…
It can even be comfy!

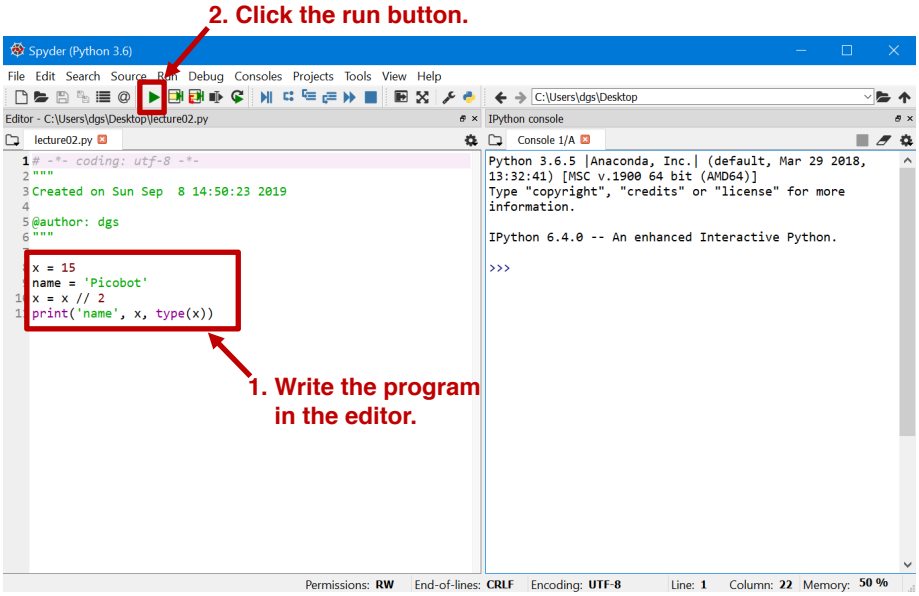
Python Programming: **console** & **Editor**

- Python includes a console and an editor for programming
- We are able to execute only one line of program in console
- The editor window which is called **integrated development environment** (IDE) makes it possible to program multiple lines of codes.
- IDE provides comprehensive facilities to computer programmers such as debugging etc…
- There are several IDEs for Python including Spyder, IDLE, PyCharm, Atom, PyDev etc…
- We will use Spyder which is a simple/free/popular IDE for Python

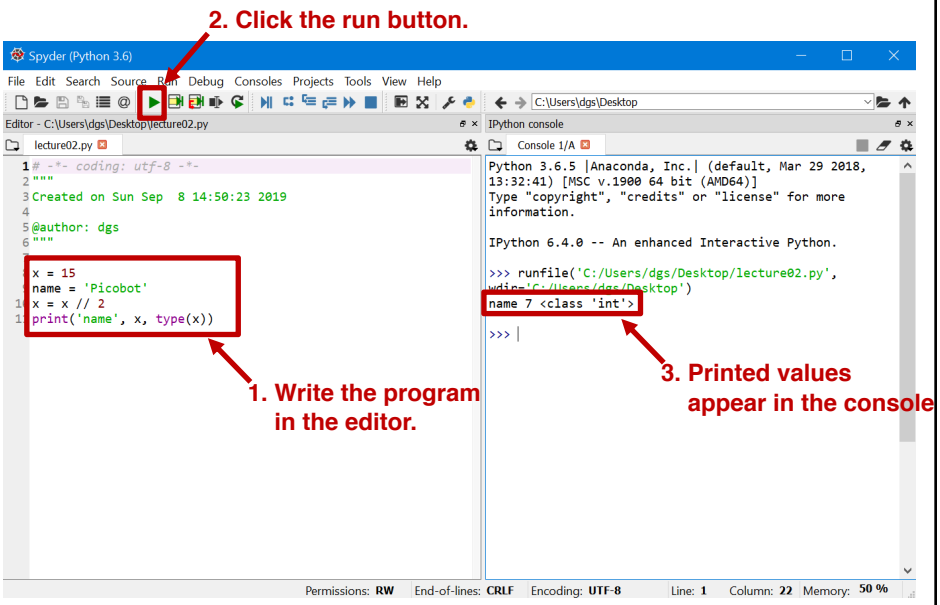
Running a Program in Spyder



Running a Program in Spyder



Running a Program in Spyder



Arithmetic in Python

- Numeric operators include:
  - `+` addition
  - `-` subtraction
  - `*` multiplication
  - `/` division
  - `**` exponentiation
  - `%` modulus: gives the remainder of a division

Arithmetic in Python (cont.)

- The operators follow the standard order of operations.
  - example: multiplication before addition
- You can use parentheses to force a different order.

Data Types

- Different kinds of values are stored and manipulated differently.
- Python *data types* include:
  - integers
    - example: `451`
  - floating-point numbers
    - numbers that include a decimal
    - example: `3.1416`

Data Types and Operators

- There are really <u>two sets</u> of numeric operators:
  - one for integers (ints)
  - one for floating-point numbers (floats)
- In most cases, the following rules apply:
  - if *at least one* of the operands is a float, the result is a float
  - if *both* of the operands are ints, the result is an int
- One exception: division!

Two Types of Division

- The / operator *always* produces a float result.
  - examples:

```
>>> 5 / 3
1.6666666666666667

>>> 6 / 3
2.0
```

Two Types of Division (cont.)

- There is a separate // operator for *integer* division.

```
>>> 6 // 3
2
```

- Integer division *discards* any fractional part of the result:

```
>>> 11 // 5
2

>>> 5 // 3
1
```

- Note that it does *not* round!

Another Data Type

- A *string* is a sequence of characters/symbols
  - surrounded by single or double quotes
  - examples: `"hello"` `'Picobot'`

Variables

- Variables allow us to store a value for later use:

```
>>> temp = 77
>>> (temp - 32) * 5 / 9
25.0
```

Expressions

- *Expressions* produce a value.
  - We *evaluate* them to obtain their value.
- They include:
  - *literals* ("hard-coded" values):

    ```
    3.1416
    'Picobot'
    ```
  - variables

    ```
    temp
    ```
  - combinations of literals, variables, and operators:

    ```
    (temp - 32) * 5 / 9
    ```

Evaluating Expressions with Variables

- When an expression includes variables, they are first replaced with their current value.
- Example:

```
(temp - 32) * 5 / 9
( 77  - 32) * 5 / 9
      45    * 5 / 9
          225   / 9
              25.0
```

Statements

- A *statement* is a command that carries out an action.
- A *program* is a sequence of statements.

```
quarters = 2
dimes = 3
nickels = 1
pennies = 4
cents = quarters*25 + dimes*10 + nickels*5 + pennies
print('you have', cents, 'cents')
```

Assignment Statements

- *Assignment statements* store a value in a variable.

```
temp = 20
```

= is known as the *assignment operator*

- General syntax:

  variable = expression

- Steps:
  1) evaluate the expression on the right-hand side of the =
  2) assign the resulting value to the variable on the left-hand side of the =
- Examples:

```
quarters = 10
quarters_val = 25 * quarters
               25 * 10
               250
```

Assignment Statements (cont.)

- We can change the value of a variable by assigning it a new value.
- Example:

Fill in the blanks!

`num1 = 100` `num2 = 120`	num1 100	num2 120
`num1 = 50`	num1	num2
`num1 = num2 * 2`	num1	num2
`num2 = 60`	num1	num2

Assignment Statements (cont.)

- We can change the value of a variable by assigning it a new value.
- Example:

`num1 = 100` `num2 = 120`	num1 100	num2 120
`num1 = 50`	num1 50	num2 120

`num1 = num2 * 2`
`120 * 2`
`240`
	num1 240	num2 120

`num2 = 60`	num1 240 The value of `num1` is unchanged!	num2 60

Assignment Statements (cont.)

- An assignment statement does <u>not</u> create a permanent relationship between variables.
- ***You can only change the value of a variable by assigning <u>it</u> a new value!***

Assignment Statements (cont.)

- A variable can appear on both sides of the assignment operator!
- Example:

Fill in the blanks!

`sum = 13` `val = 30`	sum [13]	val [30]
`sum = sum + val`	sum [ ]	val [ ]
`val = val * 2`	sum [ ]	val [ ]

Assignment Statements (cont.)

- A variable can appear on both sides of the assignment operator!
- Example:

`sum = 13` `val = 30`	sum 13	val 30

`sum = sum + val`
`13 + 30`
`43`
	sum 43	val 30

`val = val * 2`
`30 * 2`
`60`
	sum 43	val 60

Creating a Reusable Program

- Put the statements in a text file.

```
# a program to compute the value of some coins

quarters = 2       # number of quarters
dimes = 3
nickels = 1
pennies = 4

cents = quarters*25 + dimes*10 + nickels*5 + pennies
print('you have', cents, 'cents')
```

- Program file names should have the extension `.py`
  - example: `coins.py`

Print Statements

- print statements display one or more values on the screen
- Basic syntax:

  `print(`*expr*`)`

  or

  `print(`$expr_1$`, `$expr_2$`, … `$expr_n$`)`

  where each *expr* is an expression
- Steps taken when executed:
  1) the individual expression(s) are evaluated
  2) the resulting values are displayed on the same line, *separated by spaces*
- To print a blank line, omit the expressions:

  `print()`

Print Statements (cont.)

- Examples:
  - first example:

```
print('the results are:', 15 + 5, 15 - 5)
```

output: `the results are: 20 10`

(note that the quotes around the string literal are *not* printed)

  - second example:

```
cents = 89
print('you have', cents, 'cents')
```

output: `you have 89 cents`

Variables and Data Types

- The `type` function gives us the type of an expression:

```
>>> type('hello')
str
>>> print(type(5 / 2))
<class 'float'>
```

- Variables in Python do *not* have a fixed type.
  - examples:

```
>>> temp = 25.0
>>> print(type(temp))
<class 'float'>
>>> temp = 77
>>> type(temp)
int
```

How a Program Flows...

- Flow of control = order in which statements are executed
- By default, a program's statements are executed sequentially, from top to bottom.

example program

```
total = 0
num1 = 5
num2 = 10
total = num1 + num2
```

variables in memory

total		num1	
num2			

How a Program Flows...

- Flow of control = order in which statements are executed
- By default, a program's statements are executed sequentially, from top to bottom.

example program

```
total = 0
num1 = 5
num2 = 10
total = num1 + num2
        5   +  10
       15
```

variables in memory

total	15	num1	5
num2	10		

What is the output of the following program?

```
x = 15
name = 'Picobot'
x = x // 2
print('name', x, type(x))
```

A. `Picobot 7 <class 'int'>`

B. `Picobot 7.5 <class 'float'>`

C. `name 8 <class 'int'>`

D. `name 7 <class 'int'>`

E. `name 7.5 <class 'float'>`

What is the output of the following program?

```
x = 15
name = 'Picobot'
x = x // 2                    # x = x // 2
print('name', x, type(x))         15 // 2
                                      7
        'name'   7   type(7)
                         <class 'int'>
```

A. `Picobot 7 <class 'int'>`

B. `Picobot 7.5 <class 'float'>`

C. `name 8 <class 'int'>`

D. **`name 7 <class 'int'>`**

E. `name 7.5 <class 'float'>`

Extra Practice: What about this program?

```
x = 15
name = 'Picobot'
x = 7.5
print(name, 'x', type(x))
```

A. `name x <class 'float'>`

B. `Picobot 7.5 <class 'float'>`

C. `Picobot x <class 'float'>`

D. `Picobot 15 <class 'int'>`

E. `name 7.5 <class 'str'>`

Extra Practice: What about this program?

```
x = 15
name = 'Picobot'
x = 7.5
print(name, 'x', type(x))
```

```
  'Picobot' 'x'  type(7.5)
                 <class 'float'>
```

A. `name x <class 'float'>`

B. `Picobot 7.5 <class 'float'>`

C. **`Picobot x <class 'float'>`**

D. `Picobot 15 <class 'int'>`

E. `name 7.5 <class 'str'>`

What are the values of the variables after the following code runs?

```
x = 5
y = 6
x = y + 3
z = x + y
x = x + 2
```

x	y	z

On paper, make a table for the values of your variables!

	x	y	z
A.	11	6	15
B.	11	6	11
C.	11	6	17
D.	7	6	11

E. none of these, because the code has an error

What are the values of the variables after the following code runs?

```
x = 5
y = 6
x = y + 3
z = x + y
x = x + 2
    9 + 2
    11
```

x	y	z
5		
5	6	
9	6	
9	6	15
11	6	15

On paper, make a table for the values of your variables!

	x	y	z
A.	**11**	**6**	**15**
B.	11	6	11
C.	11	6	17
D.	7	6	11

E. none of these, because the code has an error

changing the value of x does *not* change the value of z!

Strings: Numbering the Characters

- The position of a character within a string is known as its *index*.
- There are two ways of numbering characters in Python:
  - from left to right, starting from 0

```
 0  1  2  3  4
'Perry'
```

  - from right to left, starting from -1

```
-5 -4 -3 -2 -1
'Perry'
```

  - `'P'` has an index of 0 or -5
  - `'y'` has an index of 4 or -1

String Operations

- Indexing: *string*[*index*]

```
>>> name = 'Picobot'
>>> name[1]
'i'
>>> name[-3]
'b'
```

- Slicing (extracting a substring): *string*[*start*:*end*]

```
>>> name[0:2]
'Pi'
>>> name[1:-1]
'icobo'
>>> name[1:]
'icobot'
>>> name[:4]
'Pico'
```

from this index

up to but ***not including*** this index

String Operations (using IPython console on Spyder)

- Indexing: *string*[*index*]

```
>>> name = 'Picobot'
>>> name[1]
result: 'i'
>>> name[-3]
result: 'b'
```

- Slicing (extracting a substring): *string*[*start*:*end*]

from this index

up to but ***not including*** this index

```
>>> name[0:2]
result: 'Pi'
>>> name[1:-1]
result: 'icobo'
>>> name[1:]
result: 'icobot'
>>> name[:4]
result: 'Pico'
```

String Operations (cont.)

- Concatenation: *string1* + *string2*

```
>>> word = 'program'
>>> plural = word + 's'
>>> plural
'programs'
```

- Duplication: *string* * *num_copies*

```
>>> 'ho!' * 3
'ho!ho!ho!'
```

- Determining the length: `len(`*string*`)`

```
>>> name = 'Perry'
>>> len(name)
5
>>> len('')   # an empty string – no characters!
0
```

What is the value of s after the following code runs?

```
s = 'abc'

s = ('d' * 3) + s

s = s[2:-2]
```

A. `'ddab'`
B. `'dab'`
C. `'dda'`
D. `'da'`
E. none of these

What is the value of s after the following code runs?

```
s = 'abc'

s = ('d' * 3) + s
      'ddd'   + 'abc'  ➔   'dddabc'

s = s[2:-2]
    'dddabc'[2:-2]
```

```
'dddabc'
 0 1 2 3 4 5
-6 -5 -4 -3 -2 -1
```

A. 'ddab'

B. 'dab'

C. 'dda'

D. **'da'**

E. none of these

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
>>> s[0:8:2]
'bso '            # note the space at the end!
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

```
>>> s[0:8:2]
'bso '            # note the space at the end!

>>> s[5:0:-1]
'notso'
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

```
>>> s[0:8:2]
'bso '            # note the space at the end!

>>> s[5:0:-1]
'notso'

>>> s[  :  :  ]   # what numbers do we need?
'viti'

>>> s[12:21:8] + s[21::3]     # what do we get?
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
      0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
>>> s[0:8:2]
'bso '              # note the space at the end!

>>> s[5:0:-1]
'notso'

>>> s[10:23:4]     # or s[10::4] or ...
'viti'

>>> s[12:21:8] + s[21::3]     # what do we get?
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston univerrsity terriers'
      0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
>>> s[0:8:2]
'bso '              # note the space at the end!

>>> s[5:0:-1]
'notso'

>>> s[10:23:4]     # or s[10::4] or ...
'viti'

>>> s[12:21:8] + s[21::3]
        'rr'
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
>>> s[0:8:2]
'bso '            # note the space at the end!

>>> s[5:0:-1]
'notso'

>>> s[10:23:4]    # or s[10::4] or ...
'viti'

>>> s[12:21:8] + s[21::3]
       'rr'     +  'rr'
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
>>> s[0:8:2]
'bso '            # note the space at the end!

>>> s[5:0:-1]
'notso'

>>> s[10:23:4]    # or s[10::4] or ...
'viti'

>>> s[12:21:8] + s[21::3]
       'rr'     +  'rr'
            'rrrr'
```

Skip-Slicing

- Slices can have a third number: *string*[*start*:*end*:*stride_length*]

```
s = 'boston university terriers'
     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
>>> s[0:8:2]
'bso '            # note the space at the end!

>>> s[5:0:-1]
'notso'

>>> s[10:23:4]    # or s[10::4] or ...
'viti'

>>> s[12:21:8] + s[21::3]
       'rr'     +  'rr'
            'rrrr'
```

What's Next

- Complete the next pre-lecture prep by 10 a.m. Wednesday
  - on Blackboard later today, along with PDFs of past lectures
- First labs are today and tomorrow.
  - Make sure that you bring your laptop to the lab
- **Problem Set 0** (***not*** the same thing as Lab 0!)
  - due Sunday by 11:59 p.m.
- ***many*** *opportunities for help!*
  - Piazza
  - office hours – see schedule on website
  - **don't send homework questions directly to me!**
- Check Blackboard/email ***frequently*** for announcements!