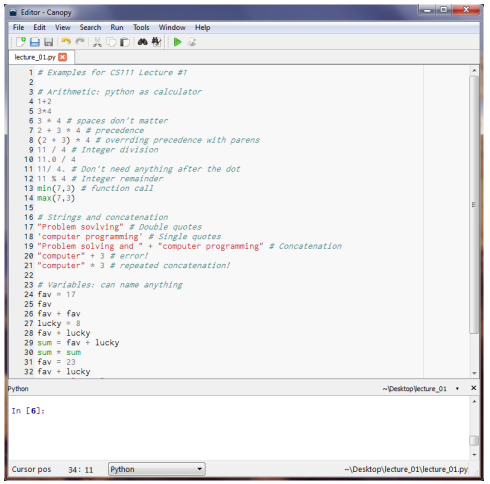
Introduction to the Python language

CS111 Computer Programming

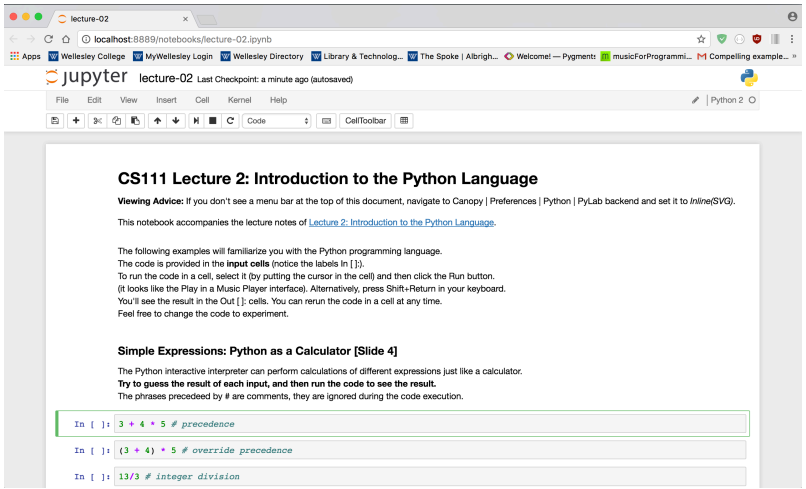
Department of Computer Science
Wellesley College

Canopy helps you edit and execute Python programs



Jupyter notebooks for hands-on activities

E.g.: lecture-02.ipynb for intro to Python



Python Intro Overview [Slide from Tuesday]

- **Values**: `10` (integer),
  `3.1415` (decimal number or float),
  `'wellesley'` (text or string)
- **Types**: numbers and text: `int`, `float`, `str`

```
type(10)
type('wellesley')
```

Knowing the **type** of a **value** allows us to choose the right **operator** when creating **expressions**.

- **Operators:** `+ - * / % =`
- **Built-in functions**: `max, min, len, int, float, str, round, print, raw_input`
- **Expressions:** (they always produce a value as a result)

```
len('abc') * 'abc' + 'def'
```

Simple Expressions: Python as calculator

Concepts in this slide: numerical values, math operators, expressions.

Input expressions
In [...]

Output Values
Out [...]

```
1+2                                                    3
3*4                                                    12
3 * 4          # Spaces don't matter                   12
3.5 * 2.0      # Floating point (decimal) operations   7.0
2 + 3 * 4      # Precedence                            14
(2 + 3) * 4    # Overriding precedence with parantheses 20
11 / 4         # Integer division                      2
11.0 / 4.0     # Floating point (decimal) division     2.75
11 / 4.0       # Floating point (decimal) division     2.75
11 % 4         # Integer remainder                     3
```


Strings and concatenation

Concepts in this slide: string values, string operators.

A string is just a sequence of characters that we write between a pair of double quotes or a pair of single quotes.

In [...] **Out [...]**

```
"CS111"          # Double quotes              'CS111'
'rocks!'         # Single quotes              'rocks!'
'CS111 ' + 'rocks!'  # Concatenation          'CS111 rocks!'
'111' + 5        # Type error                 TypeError
'111' + '5'      # Concatenation              '1115'
111 + 5          # Integer sum                116
'111' * 5        # Repeated concatenation     '111111111111111'
```


Variables

Concepts in this slide: variables, assignment statement, model.

A variable names a value that we want to use several times in a program. An assignment statement binds a name to a value, declaring in this way the new variable. A suitable **model** to think of a variable is that of a box that has a label and a value stored inside it.
Note: The symbol = is pronounced "gets" not "equals"!

In [...] **Out [...]**

```
fav = 17                  # assignment statement has no output
fav                       17
fav + fav                 34
lucky = 8
fav + lucky               25
aSum = fav + lucky
aSum * aSum               625
fav = 12
fav = fav - lucky
name = 'CS111'
name * fav                'CS111CS111CS111CS111'
```

fav ~~17~~ ~~12~~ 4

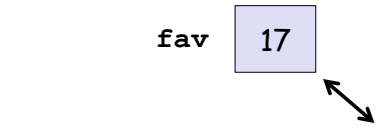
lucky 8

a_sum 25

name 'CS111'

Model: Variable as a Box

Concepts in this slide: variables, assignment statement, model, NameError




```
fav = 17   # assign

fav + 3    # lookup

# lookup and reassign
fav = fav + 3
```

- Variables are names we make up (but, there are rules for creating these names)
- A variable name should appear for the first time in an **assignment statement**.
- A value is stored in a "box".
- The variable "labels" the box.
- When a variable is used in expressions, we lookup for the "box" with that name and read its value.
- We can reassign a (new) value to a box.
- If we use a name in an expression without using it in an assignment first, we get a **NameError**.

Built-in functions: max and min

Concepts in this slide: built-in functions, arguments, function calls.

Python has many built-in functions, we don't need to define them, we just use them. Their names are shown in a green color in Canopy. Variable names are black.

In [...]	Out [...]
`min(7, 3)`	`3`
`max(7, 3)`	`7`
`min(7,3,2,9)  # can take any num. of arguments`	`2`
`smallest = min(-5, 2)  # smallest gets -5`	
`largest = max(-3, -10)  # largest gets -3`	
`max(smallest, largest, -1)`	`-1`

The inputs to a function are called its arguments and the function is said to be called on its arguments. In Python, the arguments in a function call are delimited by parentheses and separated by commas.

Built-in functions: type

Concepts in this slide: types, the function `type`.

Each Python value has a type. It can be queried with the built-in `type` function. Types are special kinds of values (not strings). Knowing the type of a value is important when writing expressions containing the value.

In [...]	Out [...]
`type(111)`	`int`
`type(4.0)`	`float`
`type('CS111')`	`str`
`type('111')`	`str`
`type(7/4)`	`int`
`type(7.0/4.0)`	`float`
`type(7.0/4)`	`float`
`type(max(7, 3))`	`int`
`x = min(7, 3)`	
`type(x)`	`int`
`phrase = 'CS111' + 'rocks!'`	
`type(phrase)`	`str`
`type(type(111))`	`type`

Built-in functions: len

Concepts in this slide: length of a string, the function `len`, TypeError

When applied to a **string**, the built-in `len` function returns the number of characters in the string. This function will throw a `TypeError` if used with non-string values.

In [...]	Out [...]
`len('CS111')`	`5`
`len('CS111 rocks!')`	`12`
`len('com' + 'puter')`	`8`
`course = 'computer programming'`	
`len(course)`	`20`
`len(111)`	`# TypeError`

Built-in functions: str

Concepts in this slide: the function `str`, complex expressions.

The `str` built-in function returns a string representation of its argument. It is used to create string values from int-s and float-s to use in expressions with other string values.

In [...]	Out [...]
`str('CS111')`	`'CS111'`
`str(17)`	`'17'`
`str(4.0)`	`'4.0'`
`'CS' + 111`	`TypeError`
`'CS' + str(111)`	`'CS111'`
`len(str(111))`	`3`
`len(str(min(17, 3)))`	`1`
`nameLen = len('CS' + str(max(110, 111)))`	
`str(nameLen)`	`'5'`

Example of a complex expression. First, max is called, then str, then +, then the function len.

Built-in functions: `int`

Concepts in this slide: the function `int`, TypeError, ValueError.

When given a string that's a sequence of digits, optionally preceded by +/-, `int` returns the corresponding integer.

When given a floating point number, `int` truncates it toward zero.

When given an integer, `int` returns that integer.

In [...]	Out [...]
`int('42')`	`42`
`int('-273')`	`-273`
`123 + '42'`	`TypeError`
`123 + int('42')`	`165`
`int('3.141')`	`ValueError`
`int('five')`	`ValueError`
`int(3.141)`	`3`
`int(98.6)`	`98`
`int(-2.978)`	`-2`
`int(42)`	`42`

Built-in functions: `float`

Concepts in this slide: the function `float`, ValueError (two different kinds)

When given a string that's a sequence of digits, optionally preceded by +/-, and optionally including one decimal point, `float` returns the corresponding floating point number.

When given an integer, `float` converts it to floating point number.

When given a floating point number, `float` returns that number.

In [...]	Out [...]
`float('3.141')`	`3.141`
`float('-273.15')`	`-273.15`
`float('3')`	`3.0`
`float('3.1.4')`	`ValueError`
`float('pi')`	`ValueError`
`float(42)`	`42.0`
`float(98.6)`	`98.6`

Oddities of floating point numbers

In computer languages, floating point numbers (numbers with decimal points) don't always behave like you might expect from mathematics. This is a consequence of their fixed-sized internal representations, which permit only approximations in many cases.

In [...]	Out [...]
`2.1 - 2.0`	`0.10000000000000009`
`2.2 - 2.0`	`0.20000000000000018`
`2.3 - 2.0`	`0.2999999999999998`
`1.3 - 1.0`	`0.30000000000000004`
`100.3 - 100.0`	`0.29999999999999716`
`10.0/3.0`	`3.3333333333333335`
`1.414*(3.14159/1.414)`	`3.1415900000000003`

Built-in functions: `round`

Concepts in this slide: the function `round`, function call with varying number of arguments.

When given **one** numeric argument, `round` returns a floating point version of the integer it's closest to.

When given **two** arguments (a numeric argument and an integer number of decimal places), `round` returns the result of rounding the first argument to the number of places specified by the second.

In [...]	Out [...]
`round(3.14156)`	`3.0`
`round(98.6)`	`99.0`
`round(-98.6)`	`-99.0`
`round(3.5)`	`4.0`
`round(4.5)`	`5.0`
`round(2.718, 2)`	`2.72`
`round(2.718, 1)`	`2.7`
`round(2.718, 0)`	`3.0`
`round(1.3 - 1.0, 1) # compare prev. slide`	`0.3`
`round(2.3 - 2.0, 1) # compare prev. slide`	`0.3`

Built-in functions: `print`

Concepts in this slide: the function `print`, an alternative way of using `print` (last line).

`print` displays a character-based representation of its argument(s) on the screen. It does not evaluate to a result value.

Input statements
In [...]

```
print(7)
print('CS111')
print('CS' + 111)
print('CS' + str(111))
print(len(str('CS111')) * min(17,3))
college = 'Wellesley'
print('I go to ' + college)
dollars = 10
print('The movie costs '
      + str(dollars) + ' dollars. '
print 1+2, 6*7, 'foo' + 'bar'
```

Characters displayed in console (*not* the output value of the expression!)

```
7
CS111
TypeError
CS111
15

I go to Wellesley

The movie costs 10
dollars.
3 42 foobar
```

* The final example shows an idiomatic use of print with commas to display multiple values on the same line. Notice the lack of parentheses.

Expression values vs. `print`

Concepts in this slide: the function `print`, `print` is different from other built-in functions.

```
In [1]: max(10,20)
Out[1]: 20

In [2]: 10 + 20
Out[2]: 30

In [3]: message = "Welcome to CS 111"

In [4]: message
Out[4]: 'Welcome to CS 111'

In [5]: print(message)
Welcome to CS 111

In [6]: print(max(10,20))
20

In [7]: print(10 + 20)
30
```

Notice the field **Out[]** when the input is a function call, expression, or variable.

The function `print` doesn't output a value, it only displays the result on the screen.

More built-in functions: `raw_input`

Concepts in this slide: the function `raw_input`, converting from string.

`raw_input` displays its argument on the screen and waits for the user to input text, followed by Enter/Return. It returns the entered value as a **string**.

```
In [1]: raw_input('Enter your name: ')
Enter your name: Phil E. Buster
Out [1]: 'Phil E. Buster'

In [2]: age = raw_input('Enter your age: ')
Enter a number: 19

In [3]: age
Out [3]: '19'

In [4]: age + 4.0
Out [4]: TypeError

In [5]: age = float(raw_input('Enter your age: '))
Enter a number: 19
In [6]: age + 4.0
Out [6]: 23.0
```

Brown text is prompt.

Magenta text is entered by user.

Variable assignment. No output.

Return value from raw_input is a **STRING**. Need to be **converted** to a numerical type as needed.

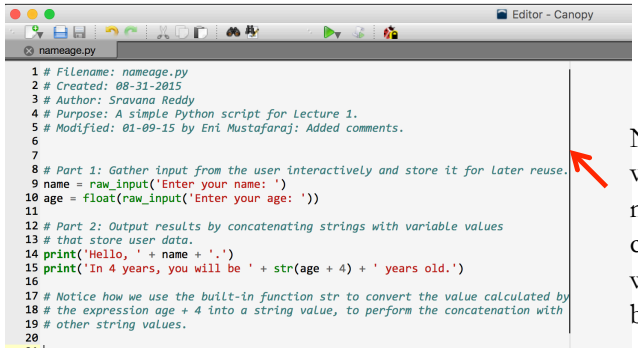
Tried to add a string and a float

Example of "nesting" two functions

Code Styling Advice

Concepts in this slide: the 80-character limit, coding advice.

The script file `nameage.py`



Notice the vertical bar that marks the 80th character. Don't write code beyond that line.

1. Give meaningful names to variables.
2. Use space around operators (e.g, =, +)
3. Use comments at the top of file
4. Organize code in "blocks" based on its meaning and provide comments.
5. Use space between blocks to improve readability.

Error messages in Python

Concepts in this slide:
Error types,
Error messages.

Type Errors

```
'111' + 5
```
TypeError: cannot concatenate 'str' and 'int' objects

```
5 + '111'
```
TypeError: unsupported operand type(s) for +: 'int' and 'str'

```
len(111)
```
TypeError: object of type 'int' has no len()

Value Errors

```
int('3.142')
```
ValueError: invalid literal for int() with base 10: '3.142'

```
float('pi')
```
ValueError: could not convert string to float: pi

Name Errors

```
CS + '111'
```
NameError: name 'CS' is not defined

Syntax Errors

```
2ndValue = 25
        ^
```
SyntaxError: invalid syntax

For syntax errors, the arrow ^ doesn't always point exactly to where the error is. In this case, the issue is the number 2 that starts the variable name. It's not allowed to start a variable name with a number.

Test your knowledge

1. Create simple **expressions** that combine **values** of different **types** and math **operators**.
2. Which operators can be used with **string values**? Give examples of expressions involving them. What happens when you use other operators?
3. Write a few **assignment statements**, using as assigned values either **literals** or expressions. Experiment with different **variable names** that start with different characters to learn what is allowed and what not.
4. Perform different **function calls** of the **built-in functions**: max, min, len, type, int, str, float, round.
5. Create **complex expressions** that combine variables, function calls, operators, and literal values.
6. Use the function **print** to display the result of expressions involving string and numerical values.
7. Write simple examples that use **raw_input** to collect values from a user and use them in simple expressions. Remember to **convert** numerical values.
8. Create situations that raise different kinds of **errors**: Type, Value, Name, or Syntax errors.