Lecture 2: Python Expressions

Acknowlegement: This notebook has been adapted adapted from the Wellesley CS111 Spring 2019 course materials (http://cs111.wellesley.edu/spring19).

1 Examples to get started in Python.

The code is provided in the **input cells** (notice the labels In [ ]:).
To run the code in a cell, select it (by putting the cursor in the cell) and then click the Run button. (it looks like the Play in a Music Player interface). Alternatively, press Shift+Return in your keyboard.
You'll see the result in the Out [ ]: cells. You can rerun the code in a cell at any time.
Feel free to change the code to experiment.

1.1 Simple Expressions: Python as a Calculator

The Python interactive interpreter can perform calculations of different expressions just like a calculator.
Try to guess the result of each input, and then run the code to see the result.
The phrases precedeed by # are comments, they are ignored during the code execution.

```
In [1]: 3 + 4 * 5 # precedence

Out[1]: 23

In [2]: (3 + 4) * 5 # parenthesis can be used to override precedence

Out[2]: 35

In [3]: 3+4*5 # spaces don't matter

Out[3]: 23

In [4]: 17/3 # floating point (decimal) division

Out[4]: 5.666666666666667

In [5]: 17//3 # integer division

Out[5]: 5

In [6]: 17 % 3 # integer remainder (% in this case is known as the modulo operator)
```

```
Out[6]: 2

In [7]: 17.0//3 # result of // is a float if either operand is a float.

Out[7]: 5.0

In [8]: 17//2.5

Out[8]: 6.0

In [9]: 17%2.5

Out[9]: 2.0
```

Summary: The results of an operator can depend on the types of the operand. For example: `7//3` returns `2` and `7.0//3` returns `2.0`; neither returns `2.3333`, but that is the result of `7/2`. Make sure to understand what is the expected value type for a simple expression.

1.1.1 Strings and Concatenation

A string is a sequence of characters that we write between a pair of double quotes or a pair of single quotes. Run every cell to see the result.

```
In [10]: "CS 134" # the string is within double quotes

Out[10]: 'CS 134'

In [11]: 'rocks!' # we can also use single quotes, it is still a string

Out[11]: 'rocks!'

In [12]: "CS 134" + 'rocks!' # example of concatenation

Out[12]: 'CS 134rocks!'
```

The above was an example of *string concatenation*, chaining two or more strings in one.
How can you fix the issue of the missing space between 111 and rocks?
Guess what will happen below:

```
In [13]: "111" + 10
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-13-bf695140c6b7> in <module>
    ----> 1 "111" + 10


        TypeError: must be str, not int
```

This is a `TypeError`, which happens when an operator is given operand values with types (e.g. `int`, `float`, `str`) that do not correspond to the expected type.

How can you fix it?

```
In [17]: '111' + '10' # or '111' + str(10)

Out[17]: '11110'
```

Repeated Concatenation: Guess the result!

```
In [18]: '123' * 4

Out[18]: '123123123123'
```

Summary: The operators + and * are the only ones you can use with values of type string. Both these operators generate concatenated strings. Be careful when using the * operator. One of the operands needs to be an integer value. Why? See what happens when you multiply two string values.

```
In [19]: 'cs' * '134'  # gives an error


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-19-95a7f66083de> in <module>
    ----> 1 'cs' * '134'  # gives an error


        TypeError: can't multiply sequence by non-int of type 'str'
```

1.1.2 Variables

A variable is essentially a box or placeholder containing a value that a programmer names or changes with an assignment statement, using =.
Variables can name any value.
Important: The symbol = is referred to as “gets” not “equals”!

```
In [20]: fav = 17 # an assignment statement has no output

In [21]: fav # this is called "variable reference" and denotes the current value of the variable

Out[21]: 17

In [22]: fav + fav # this is a simple expression that uses the current value of the variable

Out[22]: 34
```

```
In [23]: lucky = 8

In [24]: fav + lucky

Out[24]: 25

In [25]: aSum = fav + lucky # define a new variable and assign to it the value returned by the e

In [26]: aSum * aSum

Out[26]: 625
```

Let us change the value stored in the variable named `fav`.

```
In [27]: fav = 12
```

Will this change affect the variable *aSum*?
How would you check that?

```
In [28]: # No, assigning to fav does *not* change the values of previous assignments, other than
         # We can check by evaluating aSum:
         aSum

Out[28]: 25

In [29]: fav = fav - lucky # here is yet another change for the value of the variable
         # Note that the fav on the right is the current value of fav (which is 12),
         # but we're going to change the value of fav to be 12 - 8, which is 4
```

What is the current value of `fav`? How would you check that?

```
In [30]: fav

Out[30]: 4
```

1.2 Built-in Functions: `print`, `input`, `type`, `int`, `str`, `float`

`print` function will **display** characters on the screen.
Notice how we will not see the output fields labeled with `Out[]` when we use `print`.

The `input` function is used to take input from the user. By default, input value is always of type string. We can use built-in functions int and float to convert the inputed value to the desired type.

```
In [31]: print(7)

7

In [32]: print('Welcome to CS134')

Welcome to CS134
```

Using the built-in `str` function

```
In [33]: print('CS' + str(134)) # it prints the result of the expression
```

```
CS134
```

```
In [34]: college = 'Williams'
         print('I go to ' + college) # expressions can combine values and variables
```

```
I go to Williams
```

```
In [35]: dollars = 10
         print('The movie costs $' + str(dollars) + '.') # concatenation of string values
```

```
The movie costs $10.
```

When `print` is called with multiple arguments, it prints them all, separated by spaces.

```
In [36]: print(1 + 2, 6 * 7, 'CS' + '111')
```

```
3 42 CS111
```

```
In [37]: print(1,'+',2,'=',1+2)
```

```
1 + 2 = 3
```

1.3 Building interactive programs with input

```
In [38]: input('Enter your name: ') # waits for user to provide an input value and then outputs
```

```
Enter your name: Harry Potter
```

```
Out[38]: 'Harry Potter'
```

```
In [39]: age = input('Enter your age: ')  # we can store the entered input into a variable
```

```
Enter your age: 17
```

```
In [40]: age # what value is stored and of what type?
```

```
Out[40]: '17'
```

```
In [41]: type(age)
```

```
Out[41]: str
```

```
In [42]: age + 4 # will this work?


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-42-8205a21f668a> in <module>
    ----> 1 age + 4 # will this work?


        TypeError: must be str, not int


In [43]: age = int(input('Enter your age: ')) # perform conversion before storing the value

Enter your age: 17


In [44]: age + 4 # will this work now?

Out[44]: 21
```

Detour: the `type` function

```
In [45]: type(134) # this is an integer value

Out[45]: int

In [46]: type(4.0) # this is a decimal value, also known as a floating point number (because the

Out[46]: float

In [47]: type("CS134") # this is a string value

Out[47]: str

In [48]: x = "CS134 " + "rocks!"
         type(x) # we can also ask for the type of variables, the same way as for values.

Out[48]: str

In [49]: # Hey, what's the type of a type like int, float, str?
         type(int)

Out[49]: type

In [50]: # And what's the type of type?
         type(type)

Out[50]: type
```

Detour: the `int` function

```
In [51]: int('42') # convert a string value to integer

Out[51]: 42

In [52]: int('-273') # it works for negative numbers too

Out[52]: -273

In [53]: 123 + int('42')  # will this work?

Out[53]: 165

In [54]: int('3.141') # will this work?
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-54-4ca454ab1c1e> in <module>
    ----> 1 int('3.141') # will this work?


        ValueError: invalid literal for int() with base 10: '3.141'
```

```
In [55]: int('five') # will this work?
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-55-c975762f7c5a> in <module>
    ----> 1 int('five') # will this work?


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

```
In [56]: int(98.6) # convert from float to integer

Out[56]: 98

In [57]: int(-2.978) # what will this output?

Out[57]: -2

In [58]: int(422) # what will this output?

Out[58]: 422

In [59]: 64 - 4*12*1

Out[59]: 16
```

1.4 Expression values vs. `print`

In the lines below, notice what happens when you execute the cell. Notice that sometimes you see an output cell, and sometimes you don't.

```
In [60]: 20//2

Out[60]: 10

In [61]: print(20//2)

10


In [62]: 10 + 20

Out[62]: 30

In [63]: print (10 + 20)

30


In [64]: message = "Welcome to CS 134"
```

Question: why don't we see anything after executing the above cell?

```
In [65]: message

Out[65]: 'Welcome to CS 134'

In [66]: print(message)

Welcome to CS 134
```

Question: Can you notice the difference between the two lines above? Why do you think they are different?

It turns out that calling `print` returns the special `None` value. Python uses a `None` return value to indicate the function was called for its **effect** (the action it performs) rather than its **value**, so calling `print` acts like a **statement** rather than an **expression**.

To emphasize that calls to `print` act like statements rather than expressions, Canopy hides the `None` value returned by `print`, and shows no `Out[]` line. But there are situations in which the hidden `None` value can be exposed, like the following:

```
In [67]: str(print(print('CS'), print(134))) # Explain why each result line is the way it is!

CS
134
None None


Out[67]: 'None'
```

1.5 [Extra] Misc. Built-in Functions: `float`, `max`, `min`, `len`

Play with other built-in functions provided by python below.

The function `float`

```
In [72]: float('3.141') # convert a string value into a float value
```

```
Out[72]: 3.141
```

```
In [73]: float('-273.15') # it works for negative values too
```

```
Out[73]: -273.15
```

```
In [74]: float('3') # can you guess the output, why?
```

```
Out[74]: 3.0
```

```
In [75]: float('3.1.4') # what is the output for this?
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-75-b95483a60248> in <module>
    ----> 1 float('3.1.4') # what is the output for this?


        ValueError: could not convert string to float: '3.1.4'
```

```
In [76]: float('pi') # what is the output for this?
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-76-cff079c88e42> in <module>
    ----> 1 float('pi') # what is the output for this?


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

```
In [77]: float(42)   # convert from an integer to float
```

```
Out[77]: 42.0
```

The functions `max`, `min`

```
In [78]: min(7, 3)

Out[78]: 3

In [79]: max(7, 3)

Out[79]: 7

In [80]: min(7, 3, 2, 9) # notice how we can have as many arguments we want.

Out[80]: 2

In [81]: smallest = min(-5, 2) # variable smallest gets the output from the function, in this ca

In [82]: smallest # check the value stored in smallest

Out[82]: -5

In [83]: largest = max(-3, -10) # variable largest gets the value -3, which is the output of
                                # the function call with the arguments -3 and -10

In [84]: largest #check the value stored in largest

Out[84]: -3

In [85]: max(smallest, largest, -1) # we can mix variables and values as function arguments

Out[85]: -1
```

The function `len` that returns the number of characters in a string.

```
In [86]: len('CS134')

Out[86]: 5

In [87]: len('CS134 rocks!')  #try to guess before looking it up

Out[87]: 12

In [88]: len('com' + 'puter') # the expression will be evaluated first, and then the result will

Out[88]: 8

In [89]: course = 'computer programming'
         len(course)

Out[89]: 20

In [91]: len(134)  # 134 is not a string so this will result in an error
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-91-5913de64c0a5> in <module>
----> 1 len(134)  # 134 is not a string so this will result in an error


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