

CUCaTS Python Workshop - Session 2

Lent 2015

Values and Variables

Remember:

- ▶ Values are pieces of data stored in computer memory.
- ▶ Variables are labels put onto values.

```
>>> x = [7, 8]
>>> y = x
>>> y.append(11)
>>> x
[7, 8, 11]
```

Result:

- ▶ Assignment does not create a new value.
- ▶ More than one variable can label the same value.

Slicing

- ▶ Take a section of a string or list:

```
>>> name = "Satyarth"  
>>> nickname = name[0:3]  
>>> nickname  
"Sat"
```

- ▶ `[start:end]`
means "from start, up to (but not including) end".
- ▶ View the indices as between the elements:

	S		a		t		y		a		r		t		h	
0		1		2		3		4		5		6		7		8

Slicing

- ▶ start and end are optional:

```
>>> "Satyarth"[:3]  
"Sat"
```

```
>>> "Satyarth"[3:]  
"yarth"
```

- ▶ This leads to an idiom for duplicating a list or string:

```
>>> original = [1, 2, 3]  
>>> dup = original[:]  
>>> dup.append(5)  
>>> original, dup  
[1, 2, 3], [1, 2, 3, 5]
```

- ▶ Third option: step size.

```
>>> "Satyarth"[1::2]  
"ayrh"
```

```
>>> "Satyarth"[::-1]  
"htraytaS"
```

Dictionary

- ▶ Dictionaries allow us to have a bunch of **values** associated with certain **keys**.

```
ages = {"Alice": 42, "Bob": 40}
```

- ▶ One “looks up” the key and gets the value back.

```
>>> ages["Bob"]  
40
```

- ▶ Jargon: "Alice" is said to **map** to 42.

Dictionary operations

- ▶ Our example:

```
ages = {"Alice": 42, "Bob": 40}
```

- ▶ Add/update. We shall add Charlie's age and update Bob's age.

```
>>> ages["Charlie"] = 10
>>> ages["Bob"] = 41
>>> ages
{"Alice": 42, "Bob": 41, "Charlie": 10}
```

- ▶ Deleting entries

```
>>> del ages["Bob"]
>>> ages
{"Alice": 42, "Charlie": 10}
```

Dictionary methods and summary

Methods

- ▶ Getting a list of values: `list(ages.values())`
- ▶ Getting a list of keys: `list(ages.keys())`
- ▶ Checking if a key exists: `"Alice" in ages`
- ▶ Creating a new copy: `record = ages.copy()`
- ▶ We can also update or get values with `dict.get()` and `dict.update()`

Summary

- ▶ Create a dictionary: `x = {}`
- ▶ Add/update: `x[y] = z`, or `x.update({y:z})`
- ▶ Retrieve: `x[y]` or `x.get(y)`
- ▶ Delete: `del x[y]` or `x.pop(y)`

Error handling

- ▶ Looking for a key that is not in the dictionary is an error:

```
>>> ages[name] = ages[name]+1  
KeyError: "Satyarth"
```

- ▶ We can guard against it:

```
if name in ages.keys():  
    ages[name] = ages[name]+1  
  
else:  
    ages[name] = 0
```

```
try:  
    ages[name] = ages[name]+1  
  
except KeyError:  
    ages[name] = 0
```

It's easier to ask forgiveness than it is to get permission.

– Grace Hopper

Advantages of `try-except`:

- ▷ Simpler ▷ Less error prone ▷ Keeps error handling separate

- ▶ Last session we had this code:

```
def find(list, value):  
    for item in list:  
        if item == value:  
            return True  
  
    return False
```

- ▶ We wanted to change it to return the index where the element was found.

- ▶ You may have come up with something like this:

```
def find(list, value):  
    index = 0  
  
    for item in list:  
        if item == value:  
            return index  
  
        index += 1  
  
    return False
```

Enumerate

- ▶ `enumerate(list)`
- ▶ Pairs each element with its index:

```
>>> enumerate(["Alice", "Bob", "Charlie"]) *  
[(0, 'Alice'),  
 (1, 'Bob'),  
 (2, 'Charlie')]
```

- ▶ We can use it to iterate over the list:

```
def find(list, value):  
    for index, item in enumerate(list):  
        if item == value:  
            return index  
  
    return False
```

- ▶ Each loop iteration receives a pair, `(index, item)`, not just one element.

Iterables

- ▶ Not everything we iterate over with `for` is a list.
- ▶ `enumerate` does not actually give you a list!

```
>>> enumerate(["Alice", "Bob", "Charlie"])
<enumerate object at 0x005FA2B0>
>>> for item in enumerate(["Alice", "Bob", "Charlie"]):
...     print(item)
(0, 'Alice')
(1, 'Bob')
(2, 'Charlie')
```

- ▶ These are special objects which act like lists, called `iterables`.
- ▶ Can be converted to lists with `list`:

```
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Converting values

- ▶ We have seen before that if we tried to add a string to an integer, we get a `TypeError`

```
>>> x = 5
>>> y = " elephants"
>>> x + y
TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

- ▶ Similar to converting an iterable using `list`, we can use `str` to convert a value (such as an `int` or a `float`) to a string.

```
>>> str(x)
'5'
>>> str(x) + y
'5 elephants'
```

- ▶ : Tip: you can always check the type of a value with `type`:

```
>>> type(x)
<class 'int'>
>>> type(str(x))
<class 'str'>
```

Logical operators

- ▶ They operate on booleans.
- ▶ Allow you to combine expressions together.
- ▶ `and`: returns `True` if both of the operands are `True`, otherwise returns `False`.
- ▶ `or`: returns `True` if at least one of the operands is `True`, otherwise returns `False`.
- ▶ `not`: returns the complement of the operand.

Try it out!

Print a number if it is divisible by two or five:

```
for num in range(10):  
    if num % 2 == 0 or num % 5 == 0:  
        print(num)
```

Print a number if it isn't divisible by two or three:

```
for num in range(20):  
    if not (num % 2 == 0 and num % 3 == 0):  
        print(num)
```

List comprehensions

- What if we had wanted to do something further with the multiples?

```
>>> multiples = [num for num in range(20) if num % 3 == 0]
>>> multiples
[0, 3, 6, 9, 12, 15, 18]
>>> sum(multiples)
63
```

- Filters through range with a condition, including only certain elements.

Input

- ▶ Capture user input during the program.
- ▶ `input` is a function that takes a prompt as an argument and reads the input, and returning the input as a **string**.
- ▶ Example:

```
# \n just means make a new line
>>> x = input("Please enter your name\n")
Please enter your name
John Smith
>>> print(x)
John Smith
```

- ▶ Note: If you are asking for a number, remember to convert it to an integer

While

- ▶ Sometimes you want your loop to run while some condition is met, as opposed to a set number of times (`for` loops).
- ▶ Enter the `while` loop.
- ▶ Example:

```
num = 0
while not num > 0:
    num = int(input("Enter a positive number: "))
```

Breaking out of a loop

- ▶ What happens if we want to break out of our loop early?
- ▶ Example: The following code only exits if user enters 'q'

```
# This is the start of our while block
while True:
    user_input = input("Enter q to exit\n")
    if (user_input == 'q'):
        # Break out of the current loop
        break
    else:
        print("Try again")
# End of our while block
print("You've escaped my infinite loop.")
```

- ▶ Note that break only exits from the innermost block.

Continue

```
story = """
    John brought 3 apples and Jenny brought
    26 pears. Meanwhile, Bastian brought 22
    guavas!
    """

number_of_fruits = 0

for word in story.split():
    try:
        number_of_fruits += int(word)
    except ValueError:
        continue

print(number_of_fruits, "lovely fruits!")
```

Files

- ▶ Opening and reading the whole file:

```
with open("input.txt") as file:  
    print(file.read())
```

- ▶ Line by line:

```
with open("input.txt") as file:  
    for line in file:  
        print(line)
```

- ▶ Writing:

```
with open("input.txt", "w") as file:  
    file.write(str(somelist))
```

- ▶ `with` automatically closes the file at the end of the block.
Manually closing instead:

```
file = open("input.txt")  
[...]  
file.close()
```

Importing modules

We can use other people's code.

```
>>> import datetime
>>> print(datetime.date.today())
```

or...

```
>>> from datetime import date
>>> print(date.today())
```

Installing packages

Let's try installing `pygeocoder`.

We'll use `pip`, Python's package manager.

From the command line (not the interpreter):

- ▶ Windows: `py -3 -m pip install pygeocoder`
- ▶ Mac/Linux: `pip3 install pygeocoder` (add `sudo` in front if there are errors)

Exercises!

cucats.org/r/session2

Available challenges:

- ▶ Image Processing
- ▶ Caesar Cipher - Continued!
- ▶ Quickest Route Finding
- ▶ Misc exercises for this session
 - ▶ Number guessing game
 - ▶ Word count for "Pride and Prejudice"
- ▶ or continue exercises from last session

Your bible: docs.python.org/3/

Or search: devdocs.io/python/