NORTHERN ARIZONA UNIVERSITY

School of Informatics, Computing, and Cyber Systems

INF502

Lecture: Introduction to Python

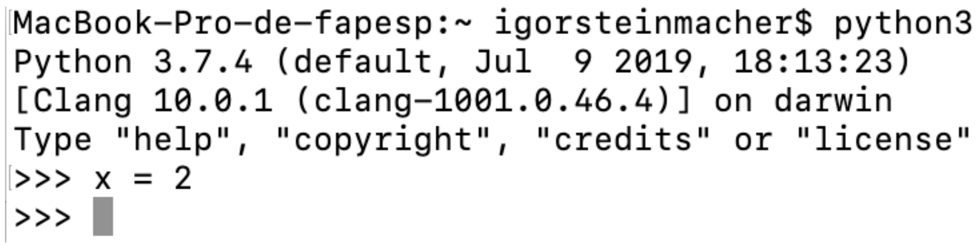
Dr. Igor Steinmacher
e-mail: Igor.Steinmacher@nau.edu

Twitter: @igorsteinmacher

Welcome to Python

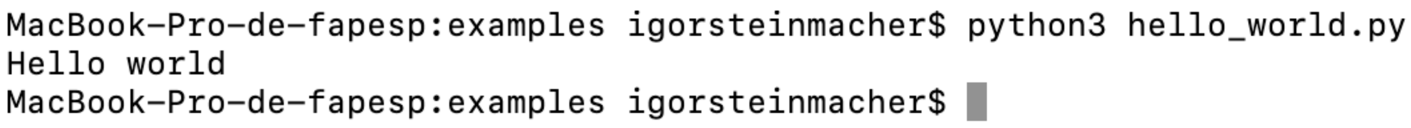
Kicking Off

- Python: Interpreted language
  - **Interactive editor**



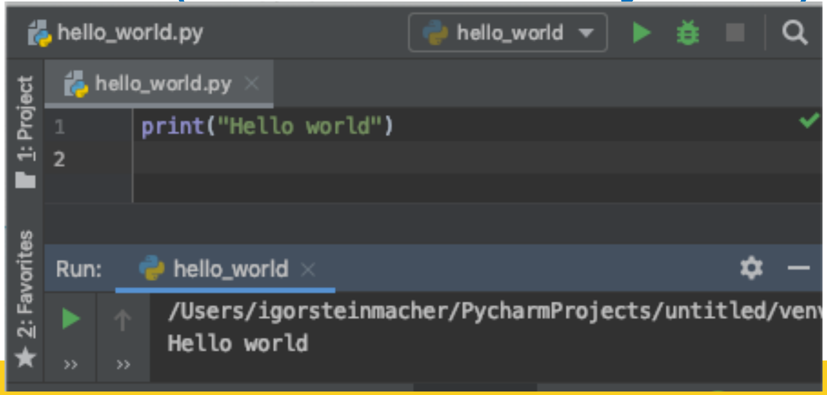
```
MacBook-Pro-de-fapesp:~ igorsteinmacher$ python3
Python 3.7.4 (default, Jul  9 2019, 18:13:23)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license"
>>> x = 2
>>> 
```

  - **Running from a file**



```
MacBook-Pro-de-fapesp:examples igorsteinmacher$ python3 hello_world.py
Hello world
MacBook-Pro-de-fapesp:examples igorsteinmacher$ 
```

  - **Using an IDE (like Anaconda or PyCharm)**



```
print("Hello world")
```

```
/Users/igorsteinmacher/PycharmProjects/untitled/venv
Hello world
```

Formatting

- Python uses **indentation** to delimit blocks of code
- Comments start with #
- Colons “:” are used to start a new block for different constructs

```python
#function that answers if a number is even or not
def isEven (number):
   #is the remainder when dividing number by 2 equals to 0
   if (number % 2 == 0):
      #yes, the number is even
      return True
   return False
```

VARIABLES

- Variables are created when they are assigned a value
  - **Type-binding is associated 'on-the-fly'**

```
>>> x = 2        #x is an integer
>>> y = 'Igor' #y is a String
>>> y = 2.5      #y is now a floating point
>>> z = [1,2,3] #z is a list
>>> #each position in z is an integer
>>> type (z)
<class 'list'>
>>> x = y = z = 4 #chained assignment
```

Arithmetic

- Mathematics apply

```
>>> a = 2+3     #a is 5
>>> b = 7-4     #b is 3
>>> c = 2*2.5   #c is 5.0 (float!!)
>>> d = 2**4.   #d is 16
>>> e = 5%2     #e is 1
>>> f = 7/2     #f is 3.5
>>> g = 7//2    #g is ...
```

STRINGS

- Strings are delimited by single or double quotation marks

```
>>> single_quote = 'python'
>>> double_quote = “python”
>>> I_want_the_quote = 'It\'s python'
>>> yes_the_quote = “It's python”
>>> multi_line = 'this is a multi line \
sentence. It is possible to break it in \
multiple lines.’
```

STRINGS

- Operations using Strings

```
#operations using strings
>>> salutation = ”Hello”
>>> name = ”John”
>>> complete_salut = salutation + ', ' + name + '!'
>>> print (complete_salut)
Hello, John!

>>> real_long_string = ('this is a long string. ',
‘It has multiple parts, ',
‘but all in one line.‘)
```

How to Input??

- Getting user inputs is usually important

```
#input() function waits for an input from the keyboard
>>> salutation = ”Hello”
>>> name = input(“Tell me your name: “)
Tell me your name: <INPUT + ENTER>
>>> complete_salut = salutation + ', ' + name + '!'
>>> print (complete_salut)
Hello, <NAME ENTERED>!

>>> weight = input(“Enter your weight in lb: ”)
Enter your weight in lb: 200
>>> weight = int(weight) #what am I doing here???
>>> weight_kg = weight/2.205
>>> print (weight_kg)
90.70294784580499
```

4-MINUTE MADNESS

- Write a program to prompt the user for hours and rate per hour to compute gross pay.
  - Enter Hours: 35
  - Enter Rate: 2.75
  - Pay: 96.25

- Write a program which prompts the user for a Celsius temperature, convert the temperature to Fahrenheit, and print out the converted temperature.

Conditional

- if – else (and more…)

```
if (x > y):
   print (“x is greater than y”)
elif (y < x):
   print (“y is greater than x”)
else:
   print (“they are equal!”)
```

```
parity = "even" if x % 2 == 0 else "odd"
```

Comparison Operations

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

Less Suffering

- Let’s use files instead of Interactive Prompt

- Create a file with the extension “.py”
  - To run:
  - python3 <filename>.py

3-MINUTE MADNESS

- **Write a program to prompt the user age and returns:**
  - **"You are a kid" → in case the person is 12 or younger**
  - **"Teenager around" → If the age is from 13 to 19**
  - **"An adult, eh?" → If older than 19**

- Do it in a file called age.py and run it

Functions

- Functions break up your code into reusable chunks
- They take arguments and can return values
  - **Increases maintainability**
  - **Makes easy to use**

```
#function that answers if a number is even or not
def isEven (number):
   #is the remainder when dividing number by 2 equals to 0
   if (number % 2 == 0):
      #yes, the number is even
      return True
   return False

isEven(10)
True

isEven(17)
False
```

Functions

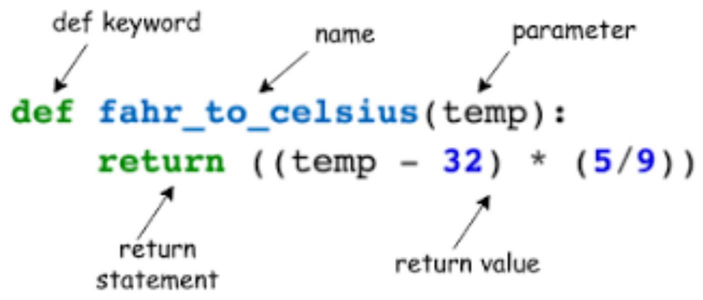
- Functions break up your code into reusable chunks
- They take arguments and can return values
  - **Increases maintainability**
  - **Makes easy to use**

```
# Calculates the body mass index (weight in kg, height in m)
def BMI(weight, height):
   bmi = weight/(height**2) #height to the power of 2
   return bmi

bmi(98, 1.85)
28.634039444850252
```

FUNCTIONS

- Two type of functions: **user-defined functions** and **user-defined**
  - `BMI(weight, height) -> Body mass index calculator`
  - `len(s) -> Return the length (the number of items) of an object.`



```
def fahr_to_celsius(temp):
    return ((temp - 32) * (5/9))
```

10-Minutes and go

- Create a function called “convert_lb_to_kg” which receives the weight in pounds and converts to kilograms

- Create a function called “convert_ft_in_to_m” which receives the height in feet and inches (2 parameters) and converts to meters

- Use the 3 functions we have now to calculate the BMI with inputs using imperial metrics.

COMBINING FUNCTIONS AND CONDITIONALS

- **Create a function to return the BMI interpretation according to CDC (centers for disease and control)**

- **The program must print one of these BMI interpretations for Adults (20 years old or older)**
  Below 18.5 = Underweight
  18.5-24.9 = Normal or Healthy Weight
  25.0-29.9 = Overweight
  30.0 or Above = Obese

LISTS

- A Python list is an **ordered**, **mutable** sequence of objects
- Lists can contain a mixture of any sort of object: numbers, Booleans, strings, other lists, …
- Lists can grow or shrink

```
>>> list_of_numbers = [1,2,3,4,5]
>>> len(list_of_numbers)
5
>>> list_of_numbers[0]
1
>>> list_of_numbers[4]
5
>>> list_of_numbers[-2]
4
```

Lists

```
#extending it
>>> list_of_numbers.extend([6,7,8])
>>> print(list_of_numbers)
[1,2,3,4,5,6,7,8]

#slicing it
>>> piece = list_of_numbers[:4] #beginning to 4
>>> print (piece)
[1,2,3,4]
>>> piece = list_of_numbers[2:6] #from position 2 to 6
>>> print (piece)
[3,4,5,6]

#shrinking it
>>> del list_of_numbers [2:5]
>>> print(list_of_numbers)
[1,2,6,7,8]
```

Lists

```
#merging
>>> list1 = [1,2,3]
>>> list2 = [4,5,6]
>>> list3 = list1 + list2
>>> print(list3)
[1,2,3,4,5,6]
>>> list1.extend(list2)
>>> print(list1)
[1,2,3,4,5,6]

#sorting
>>> list1 = [-1,4,0,9,2,7]
>>> list1.sort()
>>> print (list1)
[-1,0,2,4,7,9]
```

```
#other functions
>>> list1.append(<value>)
>>> list1.insert(<pos>,<value>)
>>> list1.remove(<value>)
>>> list1.clean()
>>> list1.remove(<value>)
>>> list1.index(<value>)
>>> list1.reverse()
>>> list1.set()
>>> <value> in list1
```

ITERATIONS (LOOPS)

- Starting with the **for** (highly used with iterable objects)

```
list_of_numbers = [1,2,3,4,5]

for element in list_of_numbers:
    print (element)
```

- But… also for counting

```
for x in range(10): #will iterate from 0 to 9
    if (x%3==0):
        continue
    print (x)

#what is the output?
```

Iterations (Loops)

- Now iterating over a list of lists

```
# grades is a list that stores the name and the grade
# of students. Each position of the list has one list
# with two positions. The first stores the name, and
# the second stores the grade

grades = [["John",60],["Paul", 84],[”Ben", 70], ["Tony",35]]
for entry in grades:
   if (entry[1]>60):
      print(entry[0] + ": approved")
   else:
      print(entry[0] + ": talk to the instructor")
```

Iterations (Loops)

- Now iterating over a list of lists

```
# grades is a list that stores the name and the grade
# of students. Each position of the list has one list
# with two positions. The first stores the name, and
# the second stores the grade

grades = [["John",60],["Paul", 84],[”Ben", 70], ["Tony",35]]
for name, grade in grades:
   if (entry[1]>60):
      print(entry[0] + ": approved")
   else:
      print(entry[0] + ": talk to the instructor")
```

ITERATIONS (LOOPS)

- Now iterating over a list of lists

```
# grades is a list that stores the name and the grade
# of students. Each position of the list has one list
# with two positions. The first stores the name, and
# the second stores the grade

grades = [["John",60],["Paul", 84],[”Ben", 70], ["Tony",35]]
for name, grade in grades:
   if (grade>60):
      print(name + ": approved")
   else:
      print(name + ": talk to the instructor")
```

ITERATIONS (LOOPS)

- Using While Loop

```
i = 1
while i <= 10:
    print(i)
    i += 1
```

Tutorial time

- I will create a program that stores a set of numbers in a list.
- The list will be filled with random numbers. The user needs to be able to:
    1. **Add a number to the list**
    2. **Present the number of elements in the list, and list the numbers (one per line)**
    3. **Clear the list**
    4. **Exit the application**

PRACTICE TIME

TUPLES

- A Python tuple is an ordered, **immutable** sequence of objects
- Like lists, but cannot be altered
  - **Do not have methods like reverse(), sort()**

```
>>> my_tuple = (6,34,6,7,2)
>>> len(mytuple)
5
>>> my_tuple[3]
7
>>> my_tuple[1:4]
[34, 6, 7]
>>> my_tuple[2]=8
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

10-MINUTE MADNESS

- Write a program which repeatedly reads integers until the user enters “done”. Once “done” is entered, print out the total, count, and average of the numbers. If the user enters anything other than an integer, print an error message and skip to the next number.
  - **Hint: isinstance (variable, int) → returns True if the variable is an integer**

- Rewrite the program above to use the following list instead of prompting the user:
  - **list_numbers=[2,6,2,5,8,2,7,3,5,3,5,7]**

Dictionaries

- A dictionary associates values with unique keys
  - **Use braces instead of square brackets**

```
>>> grades = {"John": 60, "Paul": 84, "Ben": 70, "Tony": 35}
>>> print(grades["John"])
60
>>> grades["Kate"] = 100            # adds another entry
>>> grades["John"] = 90             # changes John’s grade
>>> print(grades)
{'John': 90, 'Paul': 84, 'Ben': 70, 'Tony': 35, 'Kate': 100}
```

Dictionaries

```
>>> "John" in grades
True
>>> ”Igor" in grades
False
>>> grades.get("Joel", -1) # will return -1 (avoid errors)
>>> grades.get(”John", -1) # will return 90 (exists)

>>> all_keys = grades.keys().     # return a list of all keys
>>> all_values = grades.values() # return a list of all values
>>> all_pairs = grades.items()   # a list of (key, value) tuples
```

DICTIONARIES

```
# Nested structures... why not?
family = {"John": {"age": 30, "weight": 170, "city": "Flagstaff"},
          "Paul": {"age": 45, "weight": 200, "city": "Buenos Aires"},
          "Anna": {"age": 26, "weight": 130, "city": "Paris"}}
# Lazy iteration
for member in family:
   print(member)

for member in family:
   print(family[member])
```

Training a bit

- Given the grades dictionary provided previously

  ```
  {'John': 90, 'Paul': 84, 'Ben': 70, 'Tony': 35,
  'Kate': 100}
  ```

  Create a program that enables the user to choose one of the options:

  1. **See the grade of someone by providing a name**
  2. **Add a new student and grade**
  3. **Change a grade given a name of an existing student**
  4. **List all the grades in the dictionary**

  Each of the options needs to be implemented as a different function.

If we still have time (Extra!)

- Mountain Heights
  - **Wikipedia has a list of the tallest mountains in the world, with each mountain's elevation. Pick five mountains from this list.**
    - Create a dictionary with the mountain names as keys, and the elevations as values.
    - Create a function that receives the dictionary as a parameter and prints out just the mountains' names and elevations, as a series of statements telling how tall each mountain is: "Everest is 8848 m tall."
- Mountain Heights 2
  - **Change your function adding a second parameter (Boolean) called sorted. When this parameter is True, your algorithm needs to print the same statements as before, but in alphabetical order.**