

A Gentle Intro to Python

Nalin Ranjan | September 24, 2021



- Download the materials (slides, interactive notebook) from <https://bit.ly/3EPzAcN>. **LET US KNOW IF YOU CAN'T ASAP**
- Login to Google Colab (colab.research.google.com) and upload the `intro_to_python.ipynb` notebook. You should see the following:

▼ Python 3 Tutorial Notebook

We'll be using this notebook to follow the slides from the workshop. You can also use it to experiment with Python yourself! Simply add a cell wherever you want, type in some Python code, and see what happens!

▼ Topic 1: Printing

▼ 1.1 Printing Basics

```
[ ] # When a line begins with a '#' character, it designates a comment. This means that it's not actually a line of code
```

- Join the Sli.do:

History of Python



Why Python?

One of the easiest programming languages to learn

```
#include <stdio.h>
```

```
int main() {  
    char name[20];
```

```
    printf("Enter your name: \n");  
    scanf("%[^\n]*c", &name);
```

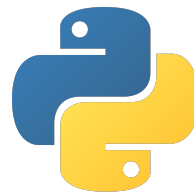
```
    printf("Your name is %s\n",  
name);
```

```
    return 0;
```

```
}
```



```
name = input("Enter your name: ")  
print("Your name is", name)
```



Why Python?

A lot of useful features already built-in for you

- Basic Data Structures
- No Low-Level Concerns (e.g. memory management)
- Rich variety of already-built libraries you can use!



Scrapy



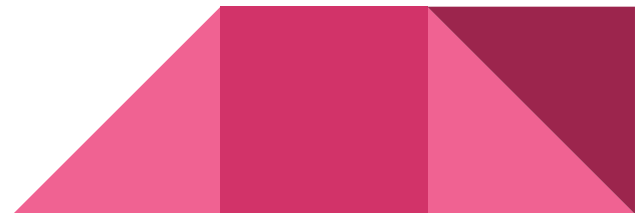
NumPy



TensorFlow



pillow



Why Python? It's used everywhere!



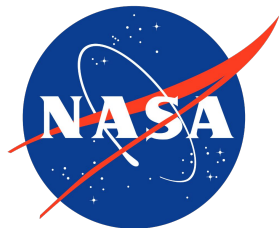
Machine Learning



Web Development



Scientific Research



How Jupyter Notebook Works

- Write code in the cells with grey background:



```
print('Welcome everyone!')  
print('My name is Nalin.')
```

- Click the play button or Shift + Enter to run your code

```
[1] print('Welcome everyone!')  
     print('My name is Nalin.')
```

```
Welcome everyone!  
My name is Nalin.
```

Simplest thing you can ask a program to do?

- Talk to you, aka print things
- In Python, this is done through the print function
- Syntax:

```
print('i am a people')  
print('hi my name', 'is', 'nalin')
```

Separate arguments with commas

```
i am a people  
hi my name is nalin
```

Notice each print statement begins on a new line

You're ready to write your first Python programs!

- Navigate to section **1.1** of the notebook and complete the two exercises (Hello World and Staircase)
- If you're done early, feel free to peruse section 1.2

- Tip: Remember that print statements start new lines by default. Can you use this when printing out the staircase?




Variables

- Often want to store data that we will later change, even if its meaning or function remains the same.

Dear **Google**,

My name is Nalin and I'm a freshman at Princeton. I saw the **Google software engineer** intern posting on Handshake and think I would be an excellent fit. I believe that my past project experience, coupled with my love for problem solving and programmatic experimentation, uniquely prepares me for this role. I would love to experience first-hand the cutting-edge things **Google software engineers** do!



Don't lie: we all do it

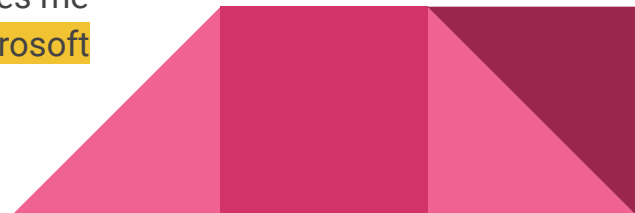
Dear **Google**,

My name is Nalin and I'm a freshman at Princeton. I saw the **Google software engineer** intern posting on Handshake and think I would be an excellent fit. I believe that my past project experience, coupled with my love for problem solving and programmatic experimentation, uniquely prepares me for this role. I would love to experience first-hand the cutting-edge things **Google software engineers** do!



Dear **Microsoft**,

My name is Nalin and I'm a freshman at Princeton. I saw the **Microsoft data scientist** intern posting on Handshake and think I would be an excellent fit. I believe that my past project experience, coupled with my love for problem solving and programmatic experimentation, uniquely prepares me for this role. I would love to experience first-hand the cutting-edge things **Microsoft data scientists** do!




Variables

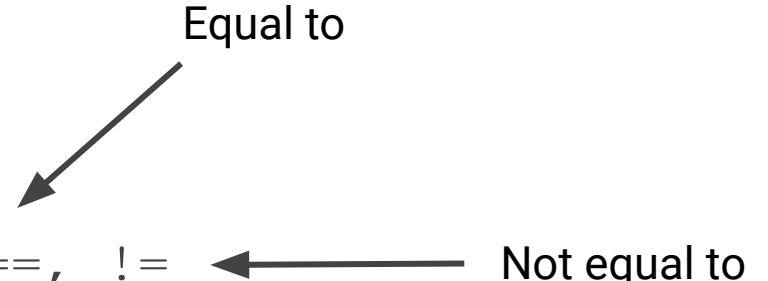
- Imagine writing a Python program that prints my cover letter
- I might want to assign variables so I only have to change the company name and role in one place:

```
company = 'Google'  
role = 'software engineer'  
  
print("Dear", company)  
...  
print("I saw the", company, role, "posting on Handshake")  
...  
print("I would love to see first-hand the cutting-edge things", company, role, "do!")
```

Numeric Variables in Python

- Integers and Decimals: 0, 3, 4.5, 2.718281828459045, -33, 1000.6
 - Can perform arithmetic operations:
 - Add: $a + b$
 - Subtract: $a - b$
 - Multiply: $a * b$
 - Float Division (returns a decimal): a / b
 - Integer Division (returns an integer): $a // b$ (return the quotient rounded down to nearest integer)
 - Modulo: $a \% b$ (remainder when a divided by b)
 - Exponentiation: $a ** b$ (a raised to b)
- 

Numeric Variables in Python (cont.)

- Don't forget the order of operations when doing arithmetic!
 - Parentheses, Exponents, Multiplication/Division/Modulo, Addition/Sub
 - We can also compare them
 - Is a less than b: $a < b$
 - Analogously for \leq , \geq , $>$, $==$, $!=$
- 

```
6 + 7 * 9 >= 70
```

```
False
```

Strings in Python

- A sequence of characters (words, codes, etc.)
- Wrapped in double quotes **or** single quotes

```
'r2d2'    "Princeton ACM"    "%&!@#!@%"    '19238'
```

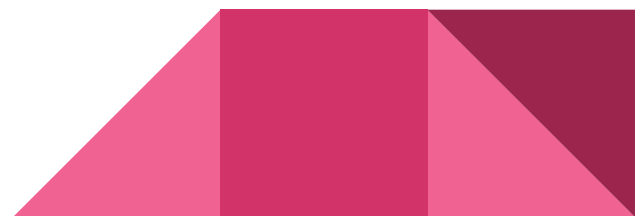
- Basic String Operations:

Concatenate: `'r2d2' + 'c3p0' --> 'r2d2c3p0'`

Count characters: `len('r2d2') --> 4`

Get kth character: `'r2d2'[2] --> 'd'`

**Very Important:
Counting starts at 0**



More Advanced String Operations

Start at first index, go up till BUT NOT INCLUDING last index

- Slicing: calculate subsequences of a string

`'hello world' [3:7]` → `'lo w'`

- Formatting: “substituting values into a formula string”

`'Name: ' + name + ', year: ' + year` → `'Name: {}, year: {}'.format(name, year)`

- `find()`: finding smaller strings inside larger ones

`'banana'.find('na')`

2

Which index in banana does the pattern 'na' first appear at, if at all?

Booleans

- Either take on the value **True** or **False**

`6 + 7 * 9 >= 70` → **False**

`'hello' != 'Hello'` → **True**

- (expression 1) **and** (expression 2): **True** only if both **True**
- (expression 1) **or** (expression 2): **True** if either is **True**
- **not** (expression): flip the value of expression
not True --> False, etc.

Mixing Types

- Be careful when working with variables of different types!

```
5 == '5'
```

False

```
age = 20  
'I am ' + age + ' years old'
```

TypeError: can only concatenate str to str

- Solution: casting

```
str(20) --> '20'    int('-20') --> -20
```

- But be careful when casting!

```
int('fish')    bool(576)
```



Check your progress

- Go to the notebook and do sections **2.1-2.3**. If you're done early, you can browse section 2.4

- Hint for the Tom Cruise one in 2.2: You're probably going to need all the string operations we discussed!



Giving computers free will

- Your thought process for deciding whether to come tonight:

If there's Tacoria → I'm coming!

No Tacoria → Meh

- Python uses **if/elif/else** statements for conditional logic

```
if boolean_condition:
```

```
    do task 1
```

```
elif other_boolean_condition:
```

```
    do task 2
```

```
else:
```

```
    do task 3
```

If boolean_condition is True, do task 1

other_boolean_condition only checked if boolean_condition was False

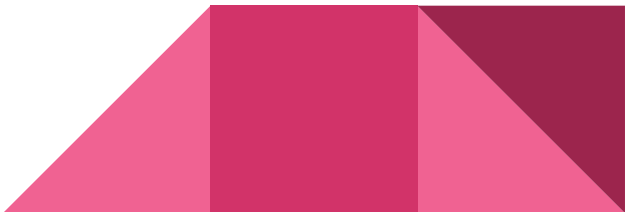
If all **if/elif blocks failed, do task 3**

Why loops?

- Often want to make programs do a lot of repetitive stuff

```
print('The square of 0 is', 0 * 0)
print('The square of 1 is', 1 * 1)
...
print('The square of 99 is', 99 * 99)
```

- We don't want to write 100 lines of repetitive code!



For loops in Python

- Iterate over all elements of a sequence (more on sequences later)

```
for i in range(0, 100):  
    print('The square of', i, 'is', i * i)
```

index of iteration

sequence of integers from 0 to 99

What we are doing every iteration of the loop

- Net effect is to print our message for every integer from 0 to 99

While loops in Python

- Keep on doing something until some condition becomes false

```
x = 0
while x < 1:
    x = x + random()
```

As long as $x < 1$ is true, keep executing the body of the loop

Add a random number between 0 and 1 to x

- Be careful with while loops! What's wrong with

```
while i < 100:
    print('The square of', i, 'is', i * i)
```

Notice the indentation!

- Indentation matters!

```
if boolean_condition:  
    do task 1  
elif other_boolean_condition:  
    do task 2  
else:  
    do task 3
```

```
x = 0  
while x < 1:  
    x = x + random()
```

```
for i in range(0, 100):  
    print('The square of', i, 'is', i * i)
```

The body of a loop or if statement is **indented** with respect to the header

Check your progress

- Complete sections **3.1** and **3.3** of the notebook. If you're done early, you can catch a glimpse of how (disgustingly) simple Python syntax can be in section 3.2

- Let any of us know if you have questions!



Sequences in Python

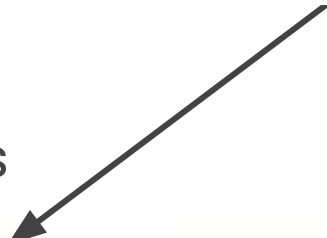
- Containers of ordered data

Strings: containers of individual characters

Tuple: comma-separated sequence of items

`(1, 2, 3, 4)` `('Nalin', 2022)` `('A', True)`

Tuples are denoted
with parentheses



List: also comma-separated sequence of items!

`[1, 2, 3, 4]` `['Nalin', 2022]` `['A', True]`

- Tuple/List difference? Lists are **mutable**, tuples aren't

Sequence operations (should look familiar!)

- Access an arbitrary element:

`[1, 2, 3][2] = 3` `'r2d2'[-1] = '2'` `(2, True)[0] = 2`

Negative indices start
at the back



- Retrieve a *slice* of the sequence:

`[1, 2, 3, 4][1:3] = [2, 3]` `'r2d2<3c3p0'[2:-3] = 'd2<c3'`

`(2, True, 'r2d2')[0:2] = (2, True)`

- Concatenate:

`[1, 2] + [3, 'pi'] = [1, 2, 3, 'pi']`



More on Slicing

- Can also slice with an increment:

[1, 2, 3, 4, 5, 6, 7, 8] [1:5:2] = [2, 4]

Start at index 1

End at (but don't include) index 5

Go in increments of 2

- If the increment is negative, the slicing goes backward:

[1, 2, 3, 4, 5, 6, 7, 8] [5:1:-2] = [6, 4]

Start at index 5

End at (but don't include) index 1

Go backward in
increments of 2

More on Slicing

- If start index omitted, Python starts at the beginning if increment is positive and end if increment is negative

[1, 2, 3, 4, 5, 6, 7, 8][:6:2] = [1, 3, 5]

Start at beginning

End at (but don't include) index 6

Go up in increments of 2

[1, 2, 3, 4, 5, 6, 7, 8][1:-2] = [8, 6, 4]

Start at end

End at (but don't include) index 1

Go backward in increments of 2

More on Slicing

- If end index omitted, Python keeps going until there are no more elements

[1, 2, 3, 4, 5, 6, 7, 8][1::2] = [2, 4, 6, 8]

Start at index 1

Go until out of bounds

Go up in increments of 2

[1, 2, 3, 4, 5, 6, 7, 8][::-2] = [8, 6, 4, 2]

Start at end


Go until out of bounds

Go backward in increments of 2

Iterating through Sequences

- Use a **for** loop and the **in** keyword:

range(4, 9)
≈ (4, 5, 6, 7, 8)



```
for number in range(4, 9):  
    # Do something for every number
```

```
for character in "hello world":  
    # Do something for every character
```

```
for item in ('random', 'tuple', True, 7):  
    # Do something for every item
```

More on lists

- **List Comprehension:** An easy way to create lists

```
[i ** 2 for i in range(5)]
```

```
[0, 1, 4, 9, 16]
```

```
week = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']  
[day for day in week if day[0] == 'S']
```

```
['Sat', 'Sun']
```

- Lists have many useful built-in functions! (see notebook for more)

Check your progress

- Complete sections **4.1** and **4.2** in the notebook

- Stand up and stretch, get water, get Tacoria, etc.



Sets in Python

- Often, we want elements in a container to be unique
- Example: Free UberEats gift cards at a finance tech talk
 - Don't want to give the person who clicks 'sign-up' 10 times 10 gift cards!

```
free_ubereats = set() # Initialize empty set s = {}
free_ubereats.add('Nalin') # s = {'Nalin'}
free_ubereats.add('Sacheth') # s = {'Sacheth', 'Nalin'}
free_ubereats.add('Nalin') # s is unchanged!
free_ubereats.remove('Sacheth') # s = {'Nalin'}
```

Some more set operations

- Initialize a non-empty set from list:

```
free_ubereats = set(['Nalin', 'Sacheth'])
```

- Union of sets: put all of the elements in either into one big set

```
free_ubereats = set(['Nalin', 'Sacheth'])  
free_grubhub = set(['Howard', 'Sacheth'])  
free_food = free_ubereats.union(free_grubhub) # {'Sacheth', 'Howard', 'Nalin'}
```

- Intersection of sets: find the set of elements that are in both sets

```
lucky_ba5tards = free_ubereats.intersection(free_grubhub) # {'Sacheth'}
```

Dictionaries in Python

- Dictionaries are just sets where each element has a value
- Like an actual dictionary!

```
{'run': 'defn of run', 'hello': 'defn of hello', 'that': 'defn of that'}
```

key/element
(must be unique)

value associated with key (doesn't have to be unique)

- Dictionary lookup is very fast!



Dictionary Example

```
favorite_ic = {} # Creates new empty dictionary
favorite_ic['Sacheth'] = 'CnC' # {'Sacheth' : 'CnC'}
favorite_ic['Nalin'] = 'Mint' # {'Sacheth' : 'CnC', 'Nalin': 'Mint'}
favorite_ic['Nalin'] = 'Cookie Dough' # {'Sacheth' : 'CnC',
                                         # 'Nalin': 'Cookie Dough'}

print(favorite_ic.keys()) # {'Nalin', 'Sacheth'}
del favorite_ic['Nalin'] # {'Sacheth': 'CnC'}
```

.keys() function returns all the keys in the dictionary

Dictionary Keys don't have to be strings! Can be tuples, numbers, or any other *immutable data type*

Sets and Dictionaries are iterable too

```
for element in some_set:  
    # do something with the element  
  
for key in dictionary:  
    # do something with the key  
  
for key, val in dictionary.items():  
    # do something with both the key and its value
```

Check your progress

- Complete section **4.3** of the Jupyter Notebook

- Ask us questions, socialize with your neighbor, etc. We're almost done!



Functions in Python

- A block of code that runs when it is called
- Why? Don't want repetitive bits of code we're going to reuse
- Anatomy of a function:

All functions start with def

Name of the function

Inputs to the function

```
def my_max(a, b):  
    if a < b: return b  
    else: return a
```

Body of the Function
(its implementation)

What the function outputs

Functions Example

```
def reverse_name(name):  
    return name[::-1]
```

```
def say_hello(name):  
    print('Hello', name)
```

```
say_hello('Nalin') # Prints 'Hello Nalin'  
reverse = reverse_name('Nalin') # reverse = 'nilaN'  
say_hello('Nalin', 'Ranjan') # ERROR: say_hello only takes one argument  
result = say_hello('Sacheth') # result = None because say_hello doesn't  
                                # return anything
```

Functions inside of functions

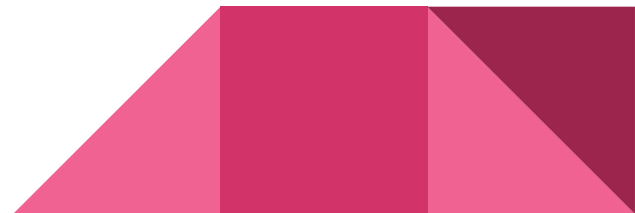
- Calling functions inside of functions is allowed!

```
def reverse_name(name):  
    say_hello(name)  
    return name[::-1]
```

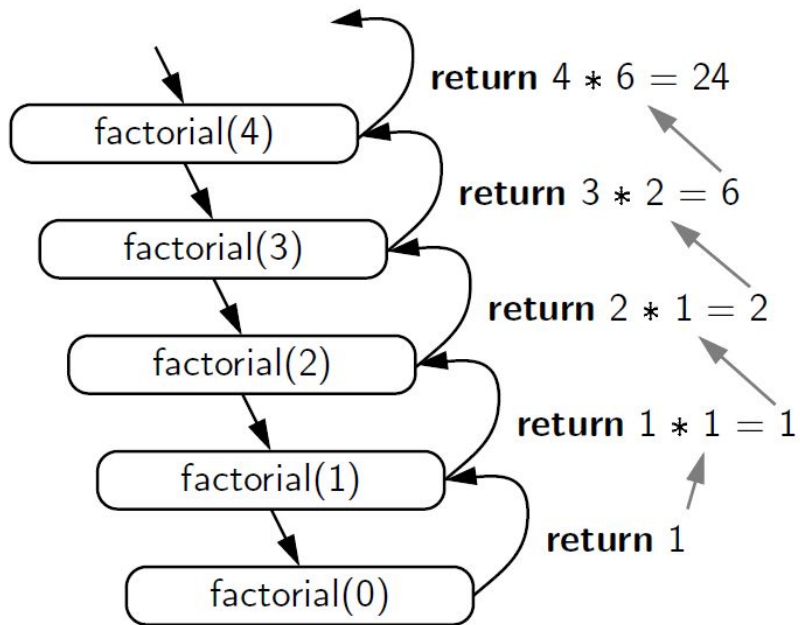


Effect: We say hello and then output their reversed name

- You can even call the function itself inside of a function! This is called **recursion**



Recursion Example: Factorial



```
def factorial(n):  
    if n == 0: return 1  
    return factorial(n - 1) * n
```

Crucial to have this
base case! Why?

Check your progress

- Complete sections **5.1** and **5.2** of the Jupyter Notebook. If you're done early, check out section **5.3**



That's it! Some last thoughts:

- Use the rich set of resources Python has to offer! Google and StackOverflow will become your best friend
- If you're doing a pretty simple task and find yourself writing a lot of code, there's probably a better way in Python – again, Google!
- If you're a COS major, don't let Python be the only language you know. Python simplifies a lot of stuff you should know about



Use us as resources!

- Ask questions now!
- Email us at ptonacm@princeton.edu with any questions!
- We'll post the slides and notebooks (including a solution notebook) on the website for you to review
- If you're interested, check out sections 6 and 7! We'll stick around to answer any questions

