Python for Data Science

Overview of Python
Why Python
Installing Python
Installing Python Modules

Overview of the course

- Assumptions:
  - We are here to learn some new skills
  - We learn new skills by doing
  - We work better with others
  - Python is important
    - It is a glue language
      - Need minimal python skills to use
    - It is interesting on its own
      - It's a modern language with interesting features
      - It's useful where-ever modules don't exist

Python

- Python is an interpreted (scripting) language
  - Source code is compiled into a bytecode representation
  - Executed by Python virtual machine (usually implemented in C or Java)
  - If performance is needed:
    - Can call C-code from Python
    - Use PyPy with Just-In-Time compilation (JIT)

Python

- Why Python:
  - Cool language
  - Extensible through modules
    - Statistics
    - Machine learning
    - Graphics

Python

- Getting Python
  - Can use bundles (anaconda)
  - For the first half: get native Python from Python.org
    - Python 2.7 stable solution (built into MacOS)
    - **<u>Python 3.9.1</u>** the version I am using
    - Important : Allow automatic path adjustments on windows
      - This are the defaults

Python

- Using Python:
  - We are going to use IDLE
    - Can create and safe scripts
    - Can interact directly in the IDE

Python 3 Modules

- Python comes with many pre-installed modules
- We need later to install some modules
  - Use Pip
    - MacOS / Linus
      - In a shell:

```
thomasschwarz@Peter-Canisius Module1 % python3.9 -m pip install matplotlib
```

    - Windows:
      - In a command window

```
py -3.9 -m pip install matplotlib
```

Why Python

- Universal, accessible language
  - Clear and simple syntax
    - Python philosophy: The frequent should be easy
    - Made for reading
    - Used for fast prototyping

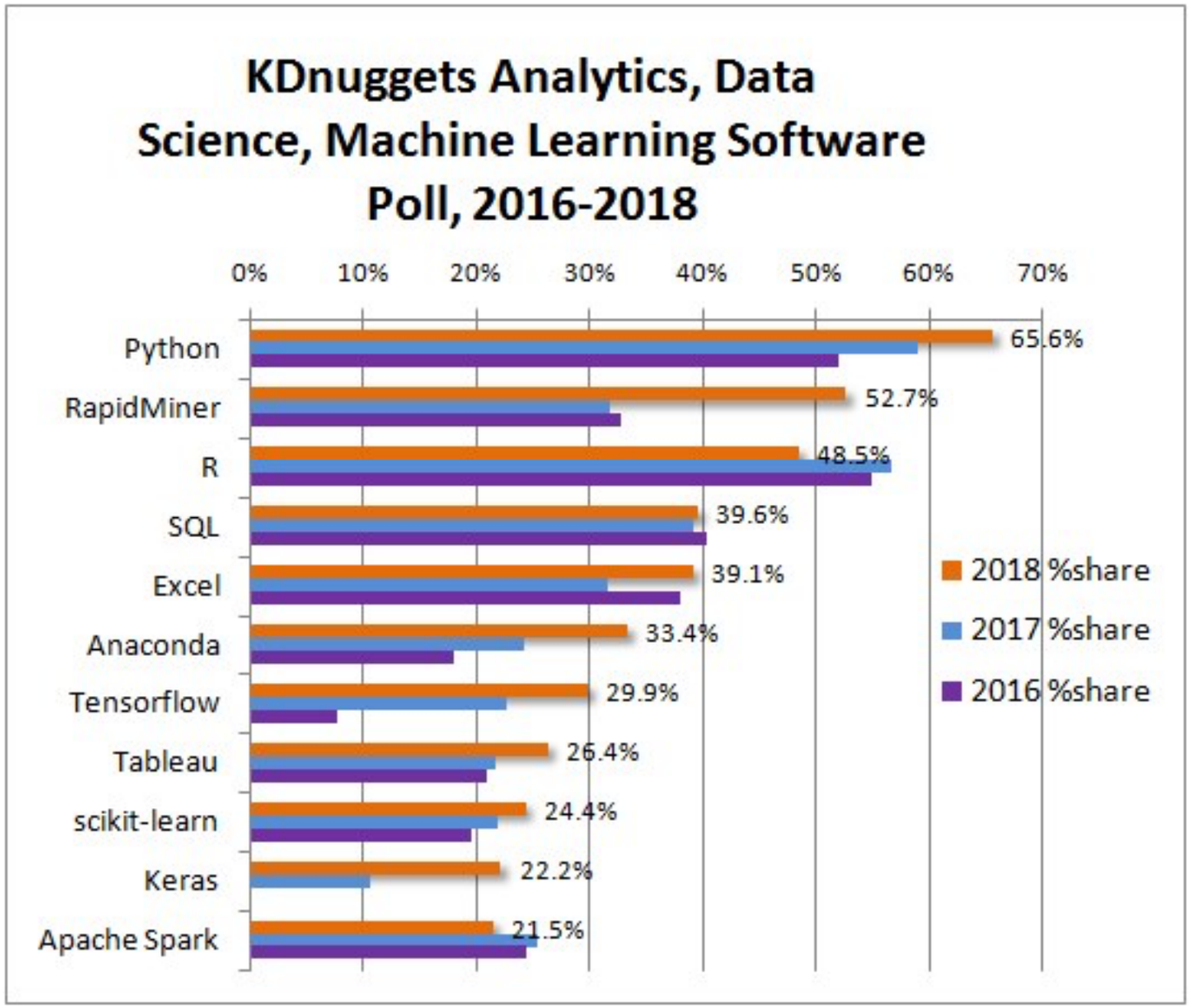
- Excellent support community
  - Help for beginners and experts is easily available

Why Python

- Universal Language
  - Serves many different constituencies
    - Examples:
      - Gaming: AI engine is usually in Python
      - String processing: Basis for digital humanities and data wrangling
      - Many extension modules
        - With scypy or numpy, fast programs for scientific programming
        - Use pyplot for good quality graphics
        - ...
      - Notebooks based on Python (Jupyter) integrate presentation, data, and programs

Why Python

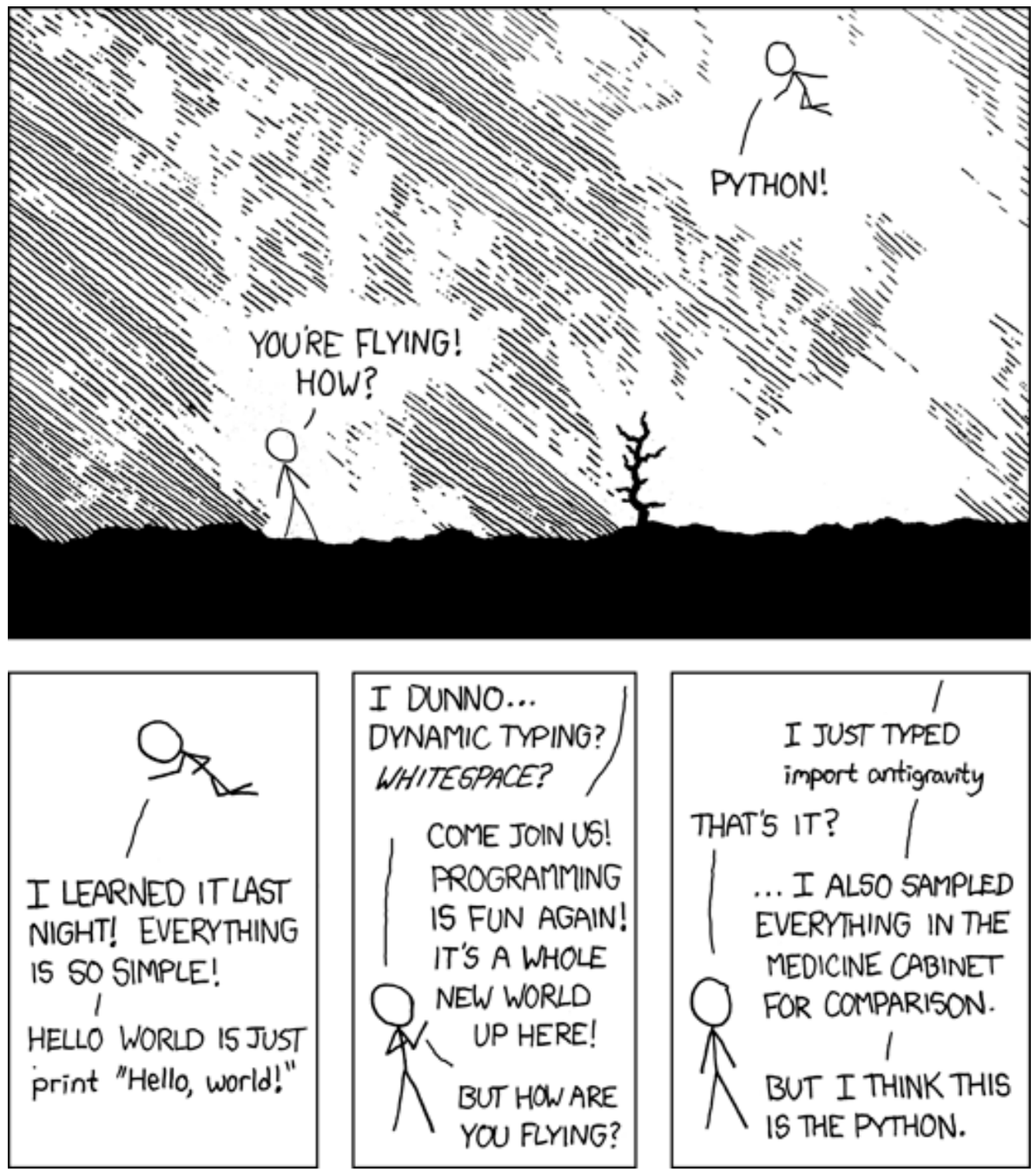
- Python in Data Science



Why Python

- https://youtu.be/pKPaHH7hnv8

Python Modules



Why Python

- Example:
  - Time series data: closing prices of four stock indices
    - given as a cvs file
  - Use Pandas in order to deal with two dimensional data
  - Use matplotlib for graphics

Why Python? Time Series Example

- Import the modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

- Import the cvs file as a pandas dataframe

```
raw_data = pd.read_csv('Index2018.csv')
values = raw_data.copy()
```

- The first column should be the index, read as a date

```
values.date = pd.to_datetime(values.date, dayfirst=
values.set_index("date", inplace = True)
print(values.describe())
print(values.head())
```

Why Python? Time Series Example

- Fill in missing values and normalize to start at 100

```
values.spx = values.spx.fillna(method = 'ffill')/values.spx['1994-01-07']*100.0
values.dax = values.dax.fillna(method = 'ffill')/values.dax['1994-01-07']*100.0
```

- Now display the US Standard & Poor and the German DAX

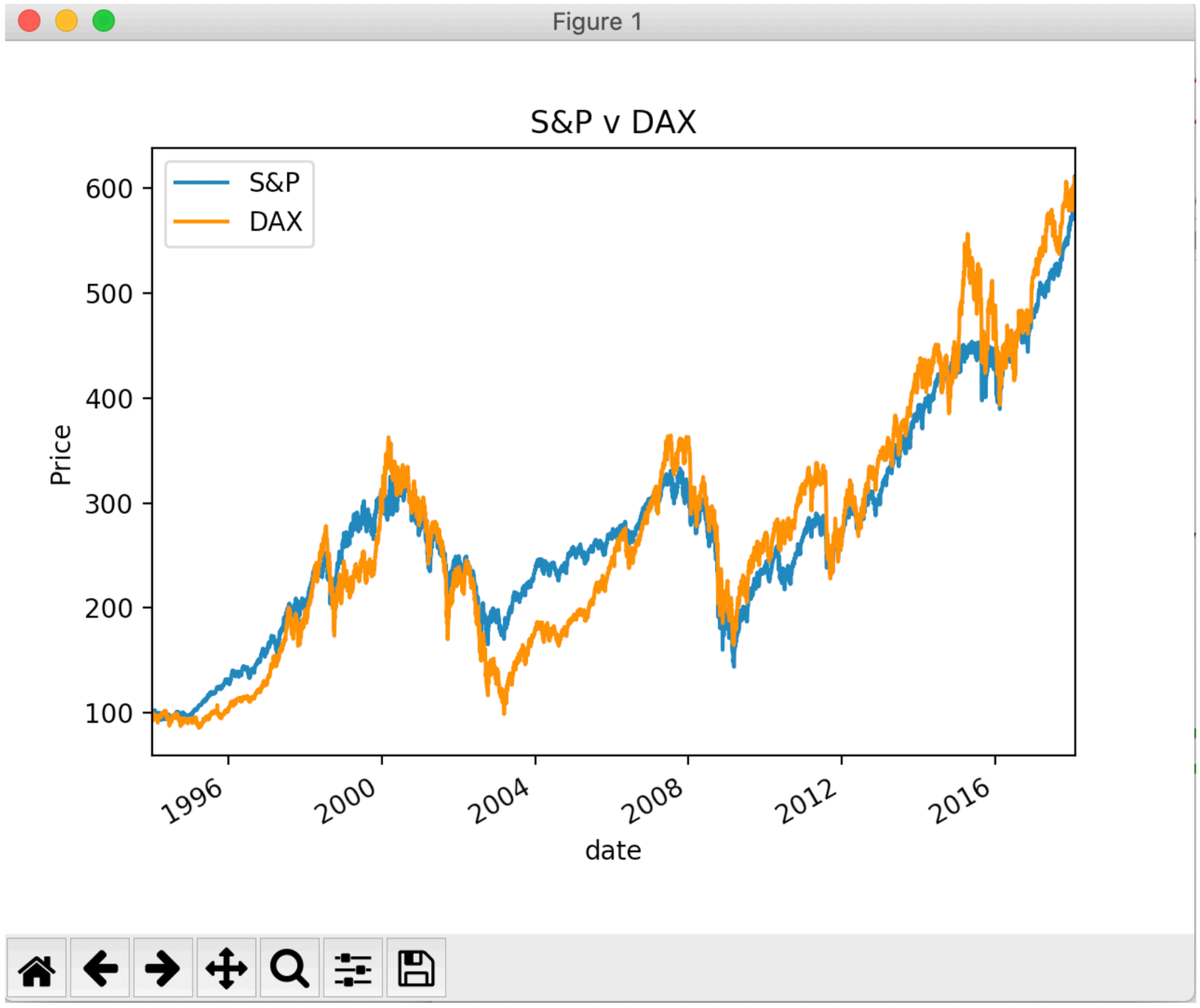
```
values.spx.plot(label='S&P')
values.dax.plot(label='DAX')
```

- Now annotate the plot and show it

```
plt.title('S&P v DAX')
plt.xlabel('date')
plt.ylabel('Price')
plt.legend()
plt.show()
```

Why Python? Time Series Example

- Result:

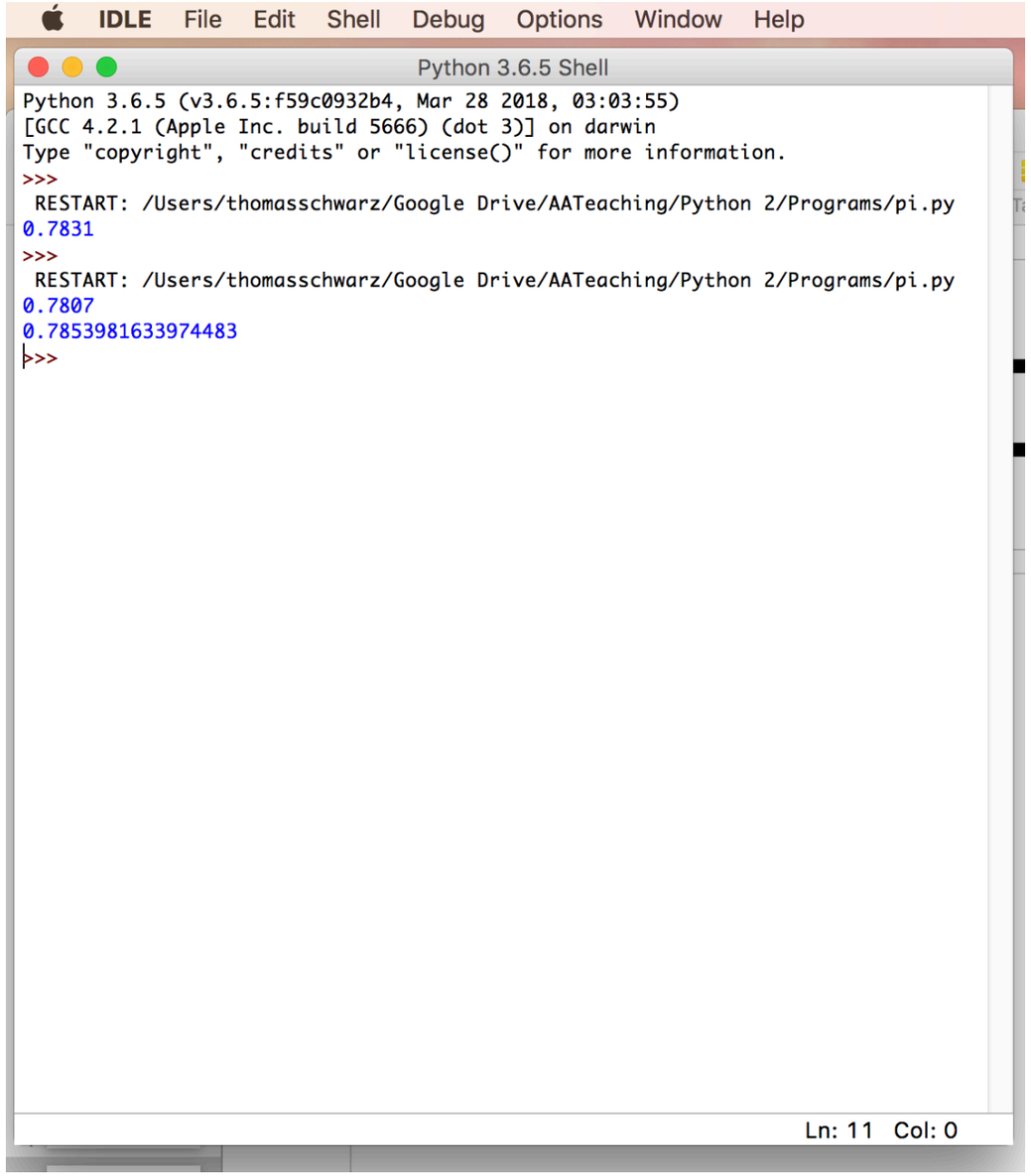


Why Python

- Most of the programming was done for us
- Needed to invoke powerful method
- Majority of the code giving to small tweaks

IDLE

- IDLE is an interactive Python interpreter
  - Can be used as a desk calculator
  - Allows you to create new files



Python Syntax

Variables and Types

- All program languages specify how data in memory locations is modified
- Python: A *variable* is a handle to a storage location
  - The storage location can store data of many types
    - Integers
    - Floating point numbers
    - Booleans
    - Strings

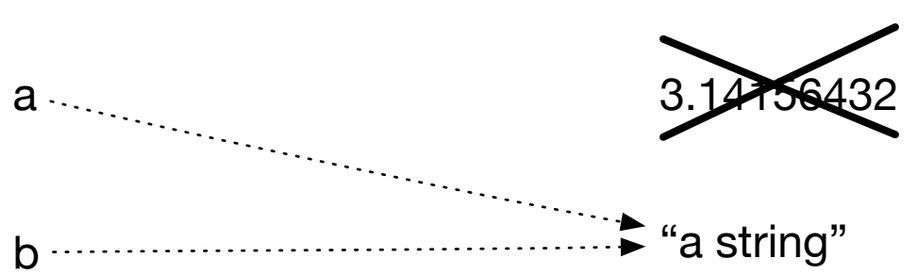
Variables and Types

- Assignment operator = makes a variable name refer to a memory location
- Variable names are not declared and can refer to any legitimate type

```
a = 3.14156432
b = "a string"
```

- Create two variables and assign values to them
- Variable *a* is of type floating point and variable *b* is of type string

```
a = b
```



- After reassigning, both variable names refer to the same value
- The floating point number is garbage collected

Expressions

- Python builds expression from smaller components just as any other programming language
  - The type of operation expressed by the same symbol depends on the type of operands
- Python follows the usual rules of precedence
  - and uses parentheses in order to express or clarify orders of precedence.

Expressions

- Arithmetic Operations between integers / floating point numbers:
  - Negation (-), Addition (+), Subtraction (-), Multiplication (*), Division (/), Exponentiation (**)
  - Integer Division //
  - Remainder (modulo operator) (%)

Expressions

- IF we use / between two integers, then we always get a floating point number
- If we use // between two integers, then we always get an integer
  - a//b is the integer equal or just below a/b

Expressions

- Strings are marked by using the single or double quotation marks
- You can use the other quotation mark within the string
- Some symbols are given as a combination of a forward slash with another symbol
  - Examples: \t for tab, \n for new line, \’ for apostrophe, \“ for double quotation mark, \\ for backward slash
  - We’ll get to know many more, but this is not the topic of today

Expressions

- Strings can be concatenated with the +
- They can be replicated by using an integer and the * sign
- Examples:
  - `“abc"+"def"  –> 'abcdef'`
  - `‘abc\"'+'fg'  –> 'abc"fg'`
  - `3*”Hi'"  –>  “Hi'Hi'Hi'"`

Change of Type

- Python allows you to convert the contents of a variable or expression to an expression with a different type but equivalent value
  - Be careful, type conversation does not always work
- To change to an integer, use `int( )`
- To change to a floating point, use `float()`
- To change to a string, use `str( )`

Example

- Input is done in Python by using the function `input`
  - Input has one variable, the prompt, which is a string
  - The result is a string, which might need to get processed by using a type conversion (aka **cast**)
  - The following prints out the double of the input (provided the user provided input is interpretable as an integer), first as a string and then as a number

```
user_input = input("Please enter a number ")
print(2*user_input)
print(2*int(user_input))
```

```
Please enter a number 23
2323
46
```

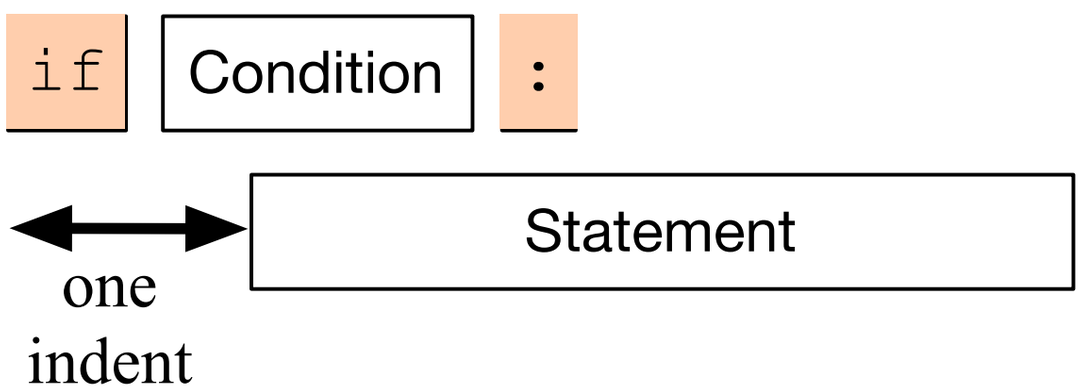
Example

- Python does not understand English (or Hindi) so giving it a number in other than symbolic form does not help
- It can easily understand “123”
- It does not complain about the expression having the same type.

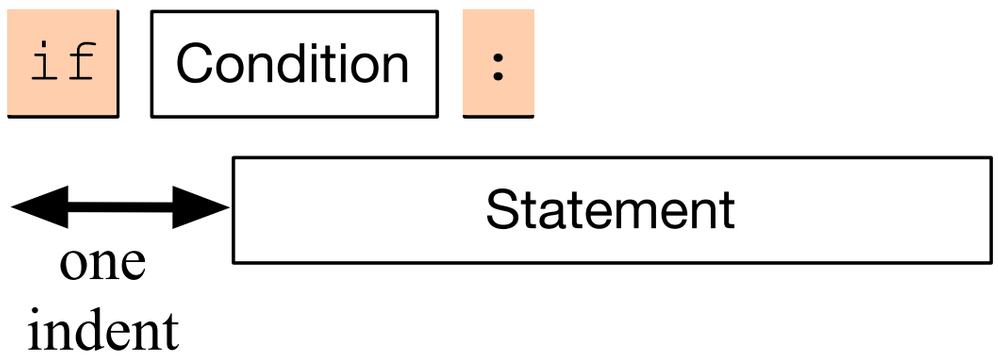
```
>>> int("two")
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    int("two")
ValueError: invalid literal for int() with base 10: 'two'
>>> float("123")
123.0
>>> int(24)
24
>>> |
```

Conditional Statements

- Sometimes a statement (or a block of statements) should only be executed if a condition is true.
- Conditional execution is implemented with the if-statement
- Form of the if-statement:



Conditional Statements



- if — is a keyword
- Condition: a Boolean, something that is either True or False
- Statement: a single or block of statements, all indented
  - Indents are tricky, you can use white spaces or tabs, but not both. Many editors convert tabs to white spaces
  - The number of positions for the indent is between 3 and 8, depending on the style that you are using. Most important, keep it consistent.

Example

p2.1.py - /Users/thomasschwarz/Googl

```
a = int(input("a number, please: "))
if a < 5:
    print("that is a small number.")
```

- First line asks user for integer input.
- Second line checks whether user input is smaller than 5.
- In this case only, the program comments on the number.

Example

p2.1.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/...

```
a = int(input("a number, please: "))
if a <= 0:
    a = -a
print("The absolute value of your number is", a)
```

- Here we calculate the absolute value of the input.
- The third line is indented.
- The fourth line is not, it is always executed.

Example

p2.1.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/...

```
a = int(input("a number, please: "))
if a <= 0:
    a = -a
    print("This is a negative number")
print("Thanks for using our program." )
```

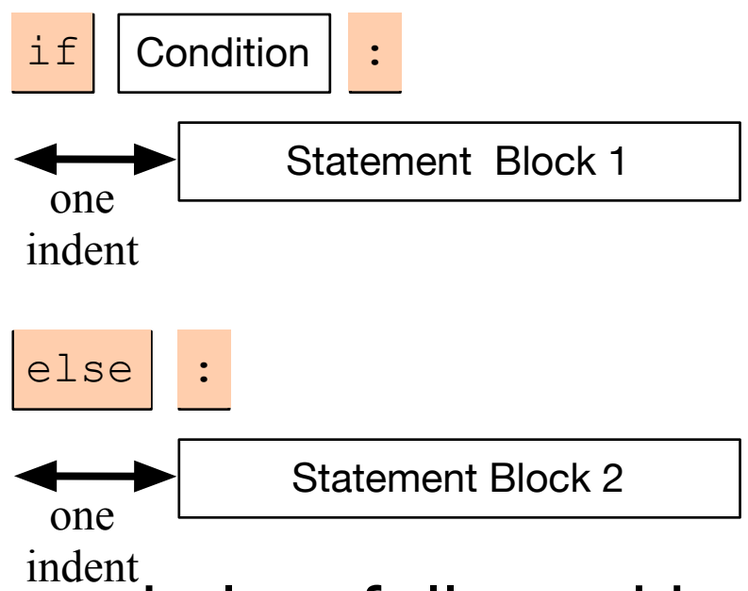
- Here, lines 3 and 4 are indented and are executed if the input is a negative integer.
- The last line, line 5, is always executed since it is not part of the if-statement

Alternative statements

- Very often, we use a condition to decide which one of several branches of execution to pursue.
- The else-statement after the indented block of an if-statement creates an alternative route through the program.

Alternative Statements

- The if-else statement has the following form:



- We add the keyword else, followed by a colon
- Then add a second set of statements, indented once
- If the condition is true, then Block 1 is executed, otherwise, Block 2.

Examples

- I can test equality by using the double = sign.
- To check whether a number *n* is even, I take the remainder modulo 2 and then compare with 0.

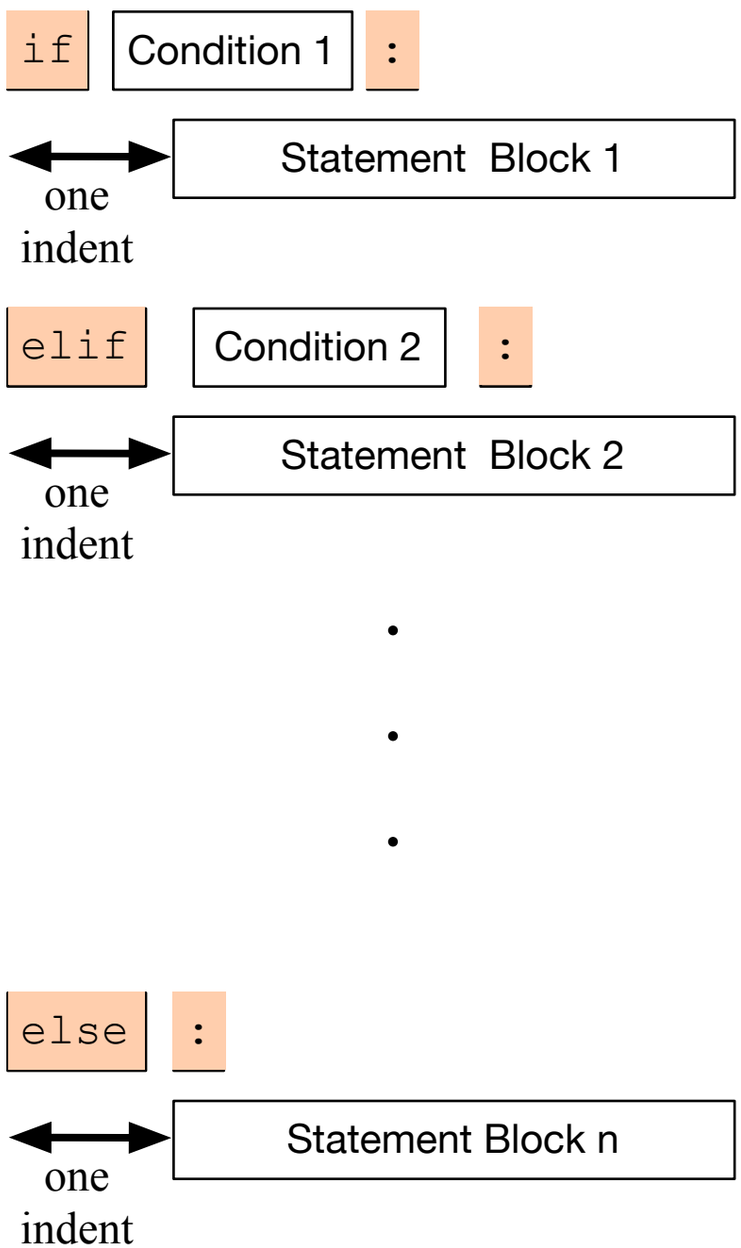
p2.2.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/...

```
number = int(input("Enter a number: "))
if number%2 ==0:
    print("The number is even.")
    print("Its square is", number**2)
else:
    print("The number is odd.")
    print("Its square-root is", number**0.5)
```

Alternative Statements

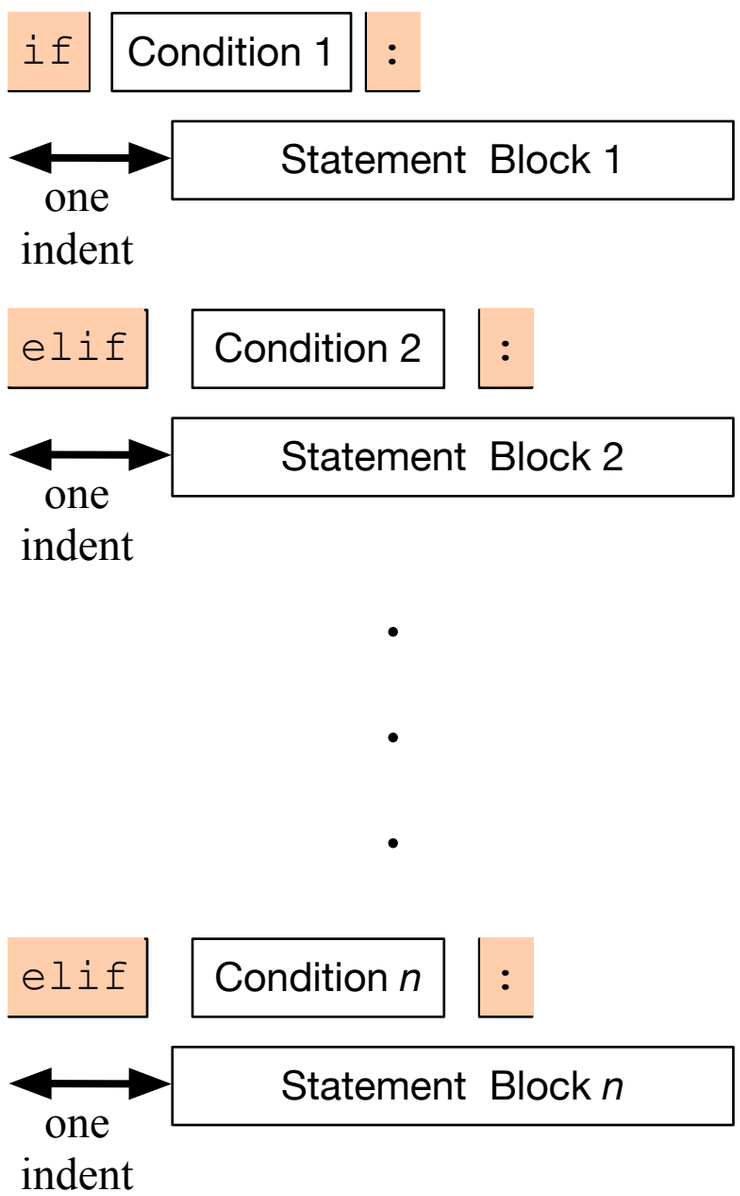
- Often, we have more than two alternative streams of execution.
- Instead of nesting if expressions, we can just use the keyword “elif”, a contraction of else if.

Alternative Statements



- One of the statement blocks is going to be executed
- The else block contains the default action, if none of the conditions are true

Alternative Statements



- Here, there is no else statement, so it is possible that none of the blocks is executed.

Examples

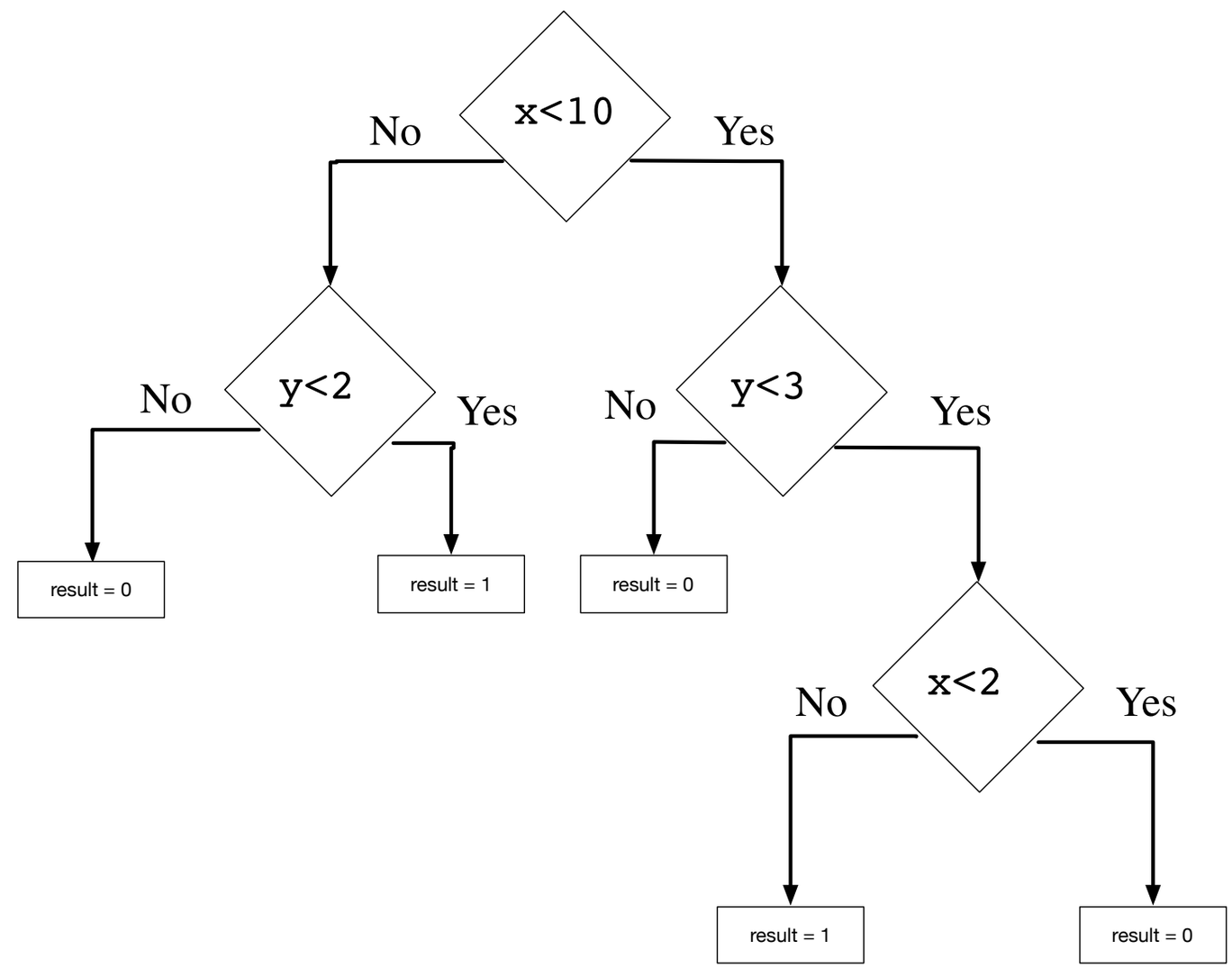
- Categorization of temperatures

```
if temperature < -25.0:
    feeling = "arctic"
elif temperature < -10.0:
    feeling = "Wisconsin in winter"
elif temperature < 0.0:
    feeling = "freezing"
elif temperature < 15.0:
    feeling = "cold"
elif temperature < 25.0:
    feeling = "comfortable"
elif temperature < 35.0:
    feeling = "hot"
elif temperature < 45.0:
    feeling = "Ahmedabad in the summer"
else:
    feeling = "hot as in hell"
```

Boolean Expressions

- Nested loops to implement decision tree:

```
if x<10:
    if y<3:
        if x<2:
            result=0
        else:
            result=1
    else:
        result=0
else:
    if y<2:
        result=1
    else result=0
```



Exercises

- Use IDLE to calculate the following expressions:
  - $\frac{2+19}{5-2} (=7)$
  - What is the remainder of 2^{20} when dividing by 5?

Exercises

- What is the result of
  - ```
    cat = '12356'
    2*cat
    ```
- and why?
    ```

Exercises

- Write a program that asks the user for an amount in Euros and converts the result to Indian Rupees.
  - As of the writing, one Euro corresponds to 87.92489314 Indian Rupees

Solution

- Open Idle and select File->New File
  - This opens up a new window
- Write your code in this window and save the file
- Then execute the Python script by using the F5 short-cut or by selecting Run->Run Module

Solution

```
user_input = input('Enter an amount in Euros: ')
euros = float(user_input)
print('The amount in Indian Rupees is ',
euros*87.92489314)
```

Loops

Thomas Schwarz, SJ

Conditions

- A condition is an expression that evaluates to True or False
- This type is called Boolean

Boolean Expressions

- The simplest Boolean expressions are `True` and `False`
- The next simplest class are numerical comparators
  - < smaller
  - > greater
  - == equals (Two! equal symbols)
  - != not equals
  - <= smaller or equal
  - >= larger or equal

```
Python 3.6.5 (v3.6.5:f59c
[GCC 4.2.1 (Apple Inc. bu
Type "copyright", "credit
>>> a = 5
>>> a !=2*2
True
>>> a != 2+3
False
>>> a<6
True
>>> a>7
False
>>>
```

Boolean Expressions

- We can combine Boolean expressions using the logical operands
  - and
  - or
  - not
- If necessary, we can add parentheses in order to specify precedence

Boolean Expression Examples

- A program that decides whether user input is divisible by 2, but not by 3.

example.py - /Users/thomasschwarz/Documents/My website/Classes/Module4/ex...

```
x = int(input("Please enter a number: "))
if x%2==0 and not x%3==0:
    print("The number is divisible by two, but not by three")
else:
    print("The number is not divisible by two or it is divisible by three.")
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/thomasschwarz/Documents/My website/Classes/Module4/example.py
Please enter a number: 5
The number is not divisible by two or it is divisible by three.
>>>
 RESTART: /Users/thomasschwarz/Documents/My website/Classes/Module4/example.py
Please enter a number: 6
The number is not divisible by two or it is divisible by three.
>>>
 RESTART: /Users/thomasschwarz/Documents/My website/Classes/Module4/example.py
Please enter a number: 4
The number is divisible by two, but not by three
>>>
```

Boolean Expression Example

- A program that checks whether the letter “a”, “A”, “e” or “E” is part of user input.
- Python allows the keyword “in” to check for the presence of letters in strings.

example2.py - /Users/thomasschwarz/Documents/My website/Classes/Module4/example2.py (3.6.5)

```
user_input = input("Please enter a string: ")
if 'a' in user_input or 'A' in user_input or "e" in user_input or "E" in user_input:
    print("present")
else:
    print("not present")
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/thomasschwarz/Documents/My website/Classes/Module4/example2.py
Please enter a string: retiuyert
present
>>>
 RESTART: /Users/thomasschwarz/Documents/My website/Classes/Module4/example2.py
Please enter a string: rtiuyirtuy
not present
>>>
```

Short-Circuit Operators

- The value of an “or”- or “and” expression is evaluated from the left to the right
  - If the first operand of an “or” is True, then the second operand is not evaluated and True is returned.
    - This is because the value of the expression is already known
  - Similarly, if the first operand of an “and” expression is False, then the second operand is not evaluated and the value of the expression is False.

Conversion of other expressions

- Any object can be tested for a truth value.
- The truth value of a non-zero number is True, otherwise False.
  - Example:

```
>>> if 5%2:
        print("5 is odd")

5 is odd
```

  - Since 5%2 evaluates to 1, it's truth value is True and the conditional statement (`print(…)`) is executed
- This behavior extends to other type of objects such as strings
  - The empty string "" has truth value 0, every other string has truth value 1.

Loops

- In CS: two types of for-loops
  - Using an index as in C, C++, Java

```
for(int i = 0; i < 10; i++)
```

  - Using lists as in Lisp

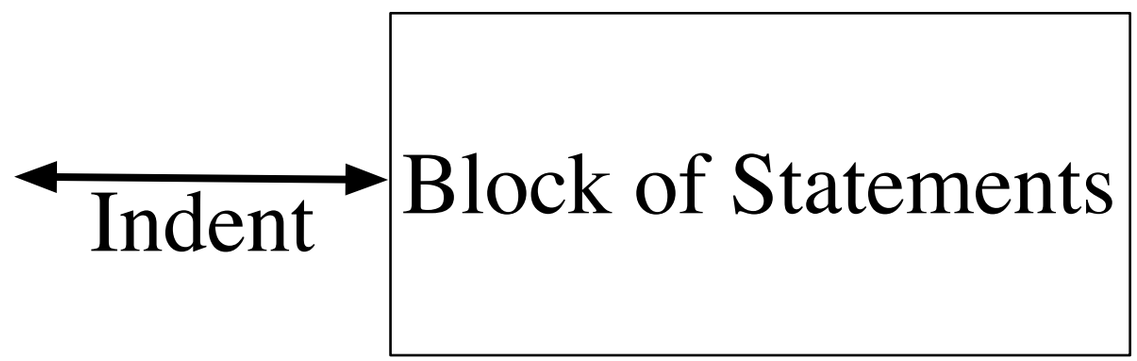
```
* (loop for x in '(a b c d e)
      do (print x) )
```

- Python for loops iterate through an 'iterator'

Loops

- To repeat a block of statements, use

```
for i in range(n):
```



Loops

- Range used to generate a list, but is now a generator
  - Like a list, but values are generated only on demand
- range with a single variable: variable is the stop value

```
range(5)                [0,1,2,3,4]
```

- range allows a start value:

```
range(2,5)              [2,3,4]
```

- range allows a stride:

```
range(2,10,3)           [2,5,8]
range(10,1,-3)          [10,7,4]
```

Loops

- Examples:
  - Calculate $\sum_{i=1}^{100} i^2 = 1^2 + 2^2 + \ldots + 99^2 + 100^2$
  - Use an accumulator to get the sum

```
def sum_of_squares(limit : int) -> int:
    accu = 0
    for i in range(1, limit+1):
        accu += i*i
    return accu
```

Notice that the sum includes 100

Loops

- Example: Count-down

```
for i in range(10, -1, -1):
    print(i)
```

```
10
9
8
7
6
5
4
3
2
1
0
```

Loops

- Calculating the factorial

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

```
accu = 1
for i in range(1, n+1):
    accu *= i
return accu
```

Calculating Sums

- For loops are handy to calculate mathematical sums
  - Geometric series:
  - Calculate $\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \ldots + \frac{1}{2^{10}}$
    - Determine iterator needs to run from 0 to 10 (inclusive)
      - `for i in range(11):`
    - Need to accumulate fractions in a sum
      - Just don't call it "sum", because "sum" has another meaning

Calculating Sums

geometric.py - /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solu...

```
accu = 0
for i in range(11):
    accu += 1/2**i
print(accu)
```

Python 3.6.5 Shell

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 03:03:55)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: /Users/thomasschwarz/Google Drive/AATeaching/Ahmedabad/Solutions/geome
tric.py
1.9990234375
>>>
```

Calculating Sums

- Admittedly, we could have used Mathematics instead
  - The sum is `1.1111111111` in binary.
  - Add `1/2**10` or `0.0000000001` in binary and we get 2.
  - Thus, the sum is `2 - 1/2**10`

Drawing Pictures

- We can use the index in a for loop in order to draw contours
  - The trick is to use string repetition instead of drawing each line separately.

for2.py - /Users/thomasschwarz/Google Drive/AATeac

```
for i in range(0,6):
    print((5-i)*" "+2*i*"*"+"*")
for i in range(5,-1,-1):
    print((5-i)*" "+2*i*"*"+"*")
```

Python 3.6.5 S

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on
Type "copyright", "credits" or "license()" for
>>>
 RESTART: /Users/thomasschwarz/Google Drive/AA
py
     *
    ***
   *****
  *******
 *********
***********
***********
 *********
  *******
   *****
    ***
     *
>>> |
```

Drawing Pictures

for3.py - /Users/thomasschwarz/Google Drive/AATeaching

```
for i in range(8):
    for j in range(2):
        print(4*(4*" "+4*"*"))
    for j in range(2):
        print(4*(4*"*"+4*" "))
```

Python 3.6.5 Shell

```
   ***   ***   ***   ***
   ***   ***   ***   ***
***   ***   ***   ***
***   ***   ***   ***
>>>
 RESTART: /Users/thomasschwarz/Google Drive/AATeachi
py
    ****    ****    ****    ****
    ****    ****    ****    ****
****    ****    ****    ****
****    ****    ****    ****
    ****    ****    ****    ****
    ****    ****    ****    ****
****    ****    ****    ****
****    ****    ****    ****
    ****    ****    ****    ****
    ****    ****    ****    ****
****    ****    ****    ****
****    ****    ****    ****
    ****    ****    ****    ****
    ****    ****    ****    ****
****    ****    ****    ****
****    ****    ****    ****
    ****    ****    ****    ****
    ****    ****    ****    ****
****    ****    ****    ****
****    ****    ****    ****
    ****    ****    ****    ****
    ****    ****    ****    ****
****    ****    ****    ****
****    ****    ****    ****
>>>
```

While Loops

- Form of the while loop:

`while` condition `:`

⟷ Statement Block

Indent

  - Keyword is while
  - Condition needs to evaluate to either True or False
    - Condition is a <u>boolean</u>

While Loop Conditions

- Statement block is executed as long as condition is valid.
  - Allows the possibility of infinite loops

Apple Inc.
One Infinite Loop
Cupertino, CA 95014
(408) 606-5775

```
while condition :
```

Statement Block

An Infinite Loop

```
while True:
  print(“Hello World”)
```

If this happens to you, you might have to kill Idle process.

While Loops can emulate for loops

- Find an equivalent while loop for the following for-loop
  - (which calculates $\sum_{\nu=1}^{n} \frac{1}{\nu}$)

```
n = int(input("Enter n: "))
suma = 0
for i in range(1,n+1):
    suma += 1/i
print("The", n, "th harmonic number is", sum)
```

While loops can emulate for loops

- Solution: the loop-variable *i* has to start out as 1 and then needs to be incremented for every loop iteration
- We stop the loop when *i* reaches *n*+1, i.e. we continue as long as *i* <= *n.*

```
n = int(input("Enter n: "))
sum = 0
i = 1
while i<= n:
    sum += 1/i
    i += 1
print("The", n, "th harmonic number is", sum)
```

Harmonic Numbers

- The *n*th harmonic number is $$h_n = \sum_{\nu=1}^{n} \frac{1}{\nu}$$
  - It is known that this series diverges.
- Given a positive number *x*, we want to determine *n* such that the *n*th harmonic number is just above x

$$\min(\{n \mid h_n > x\})$$

- Solution: add $\frac{1}{\nu}$ while you have not reached *x*

Harmonic Numbers

```
x = float(input("Enter x: "))
nu = 1
sum = 0
while sum <= x:
    sum += 1/nu
    nu += 1
print("The number you are looking for is ", nu-1,
      "and incidentally, h_n =“, sum)
```

- When we stop, we need to undo the last increment of nu, but not for sum.

Breaking out of a while loop

- You break out of a while loop, if the condition in the while loop is False
- Or by using a statement
  - `break` breaks out of the current loop
  - Can be used in for loops as well
- A related statement is the continue statement
  - `continue` breaks out of the current iteration of the loop and goes to the next
- We'll learn them in the course of the classes.

Example

- Find a number that fulfills the following congruences

$$x \equiv 2 \pmod 3$$
$$x \equiv 3 \pmod 5$$
$$x \equiv 2 \pmod 7$$

- This is Sun-Tsu's problem and the Chinese Remaindering Theorem in Mathematics helps with solving these problems.

Example

- We try out all numbers between 1 and $3 \times 5 \times 7$
  - We check each number whether they fulfill the congruences
  - If we find one, we print it out and break out of the while loop.

```
x = 1
while x < 3*5*7:
    if x%3==2 and x%5==3 and x%7==2:
        print(x)
        break
    x += 1
```

While Loops

- break: stop the execution of the loop
- continue: stop the execution of the current iteration and go back to the evaluation of the loop condition
- (Stupid) Example: Print out all even numbers from 1 to 100

```
for i in range(1, 101):
    if i%2==1:
        continue
    print(i)
```

While Loops

- A frequent pattern:
  - Have an infinite while loop
  - Break out if a certain condition is true

While Loops

- Else clause (an example that Python is not perfect)
  - Executed if a break is <u>not</u> taken

```
while condition :
        break
else :
```

While Loops

- Else clause example:

```
for n in [2,3,4,5,6,7,8,20,21,22,23,24]:
        for p in range(2, n):
            if p*(n//p) == n:  # p devides n
                print(n,'=', p, '*', n//p)
                break
        else:
            print(n, 'is prime')
```

- Notice: 'else' belongs to the inner for, not the if statement

Exercises

- Use finer and finer sums in order to calculate
$$\int_0^1 x^3 dx = \frac{1}{4}$$

Solution

- We divide the interval [0,1] into N subintervals of size 1/N
- The minimum of the function in the subinterval $\left[\frac{i}{N}, \frac{i+1}{N}\right]$ is $\frac{i^3}{N^3}$
- We multiply this with the length of the subinterval $\frac{1}{N}$ and add up to get
  - $\sum_{i=0}^{N-1} \frac{i^3}{N^3} \frac{1}{N}$
- as a lower estimate for the integral.

Solution

```
N = 10000000

suma = 0
for i in range(N):
    suma += i**3/N**4

print(suma)
```