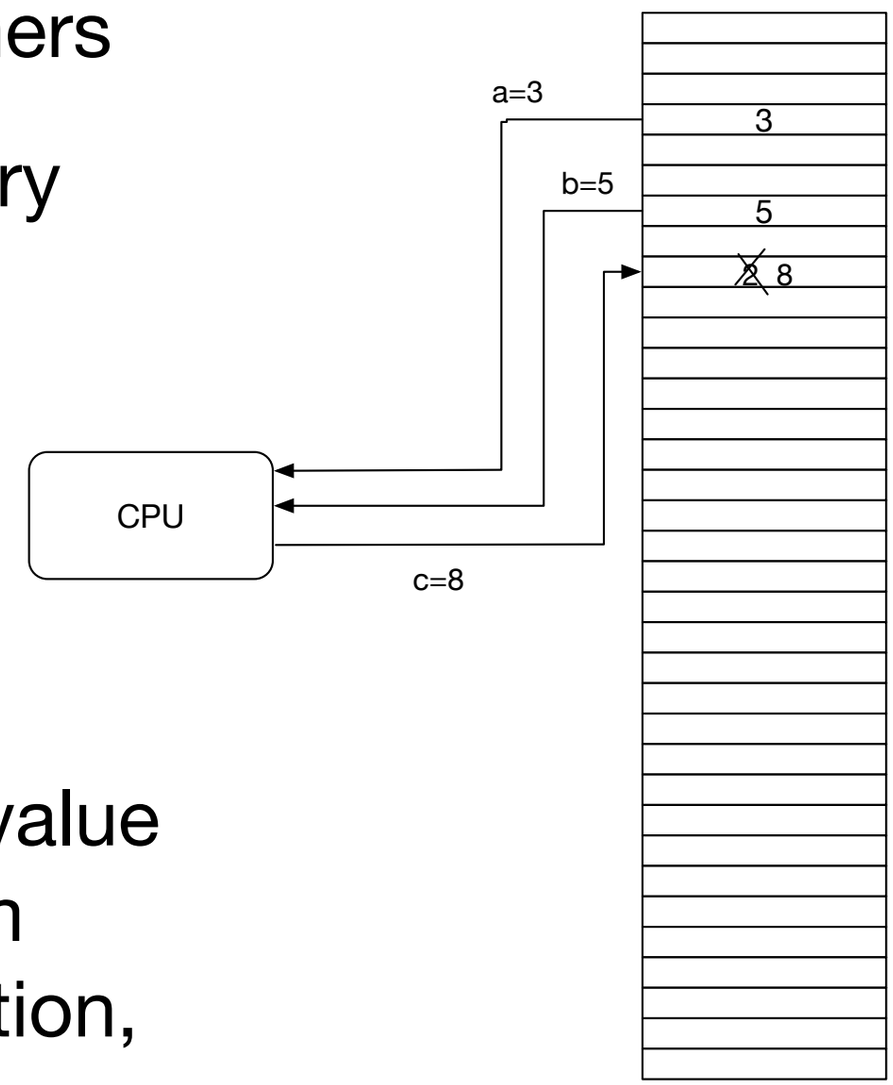
Module 2: Loops

Thomas Schwarz, SJ

Repetition

- Computational model for kindergardeners
  - We have a very large array of memory locations
  - The memory locations are variables
  - A program consists of a series of instructions
  - A typical instruction c=a+b takes a value from storage location a, a value from storage location b, does a computation, and stores in storage location c



Repetition

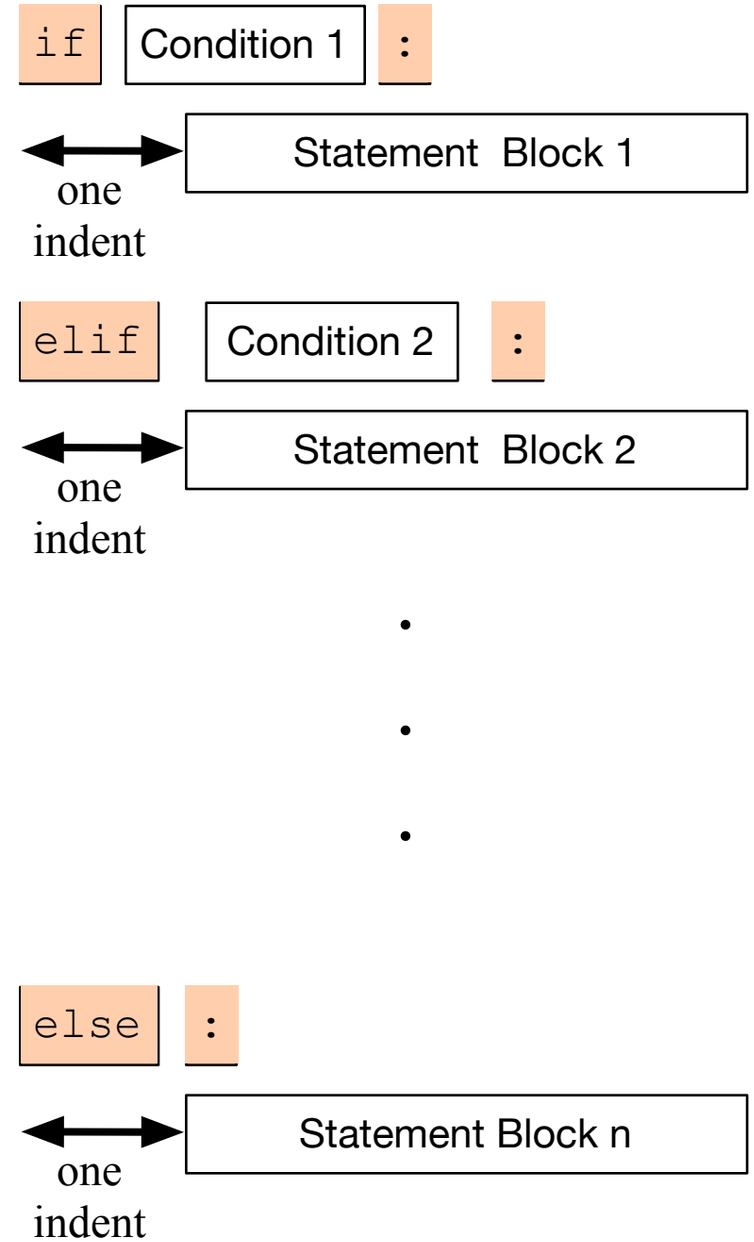
- Python variables are defined by assignment
  - They are "strongly typed":
    - E.g.: Operations depend on the type
      - + between numbers: addition
      - between strings: concatenation: `'नमस्ते'+' '+'दुनिया'`
      - * between numbers: multiplication, between integer and string:
- The same variable name can refer to entities of different types during the lifetime of a program

Repetition

- Assignment: "="
  - `a = 3*b/c`
- Operators:
  - Usually set: +, -, *, /, **
  - Binary operators: ^, |, <<, >>, &, ~
  - Unusual: // is integer division, % modulo operator

Repetition

- Conditional statements
  - if, if else, if elif … else
- Unusual:
  - White spaces form blocks
  - No parenthesis around conditions



Repetition

- Example: (Python has no switch statement)

```
if temperature < -20:
    print('welcome to Minnesota in the winter')
elif temperature < -10:
    print('I love Milwaukee in the winter')
elif temperature < 0:
    print('be careful about driving')
elif temperature < 10:
    print('Finally spring in Milwaukee')
elif temperature < 20:
    print("It's getting hot")
elif temperature < 30:
    print('normal')
elif temperature < 45:
    print('when does monsoon start')
elif temperature < 55:
    print("it's hot even for Ahmedabad")
else:
    print('where are you living')
```

Repetition

- Python strings
  - Python is very flexible about the encoding that you use
  - Python-3 scripts should be written in utf-8
  - Strings can be denoted by single or double quotation marks
    - Python is very good at interpreting what you mean but sometimes escapes are necessary

`Pep-8 style guidelines:  https://www.python.org/dev/peps/pep-0008/`

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

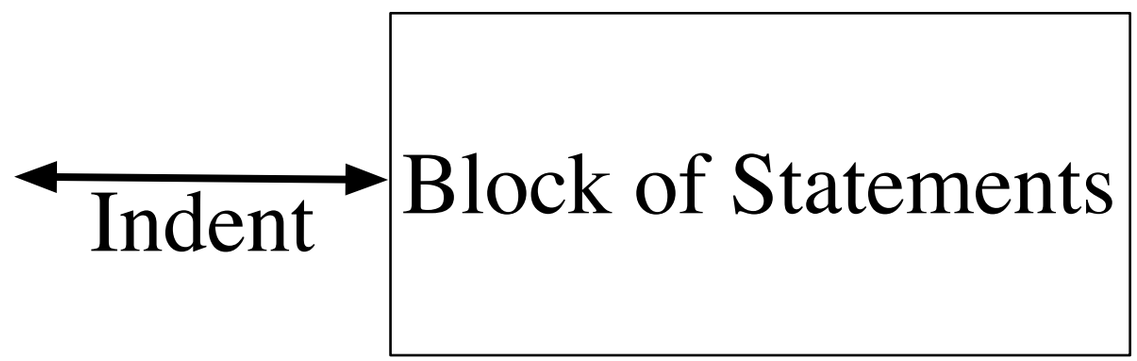
```
* (loop for x in '(a b c d e)
      do (print x) )
```

- 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 :
```

Indent ⟷ 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