Midterm 1 Preparation 1

Highlights from what we have seen so far including a list of typical errors to avoid

Variables, Operations, Types

- Every variable in Python has a type
  - But the type is not declared
  - The same variable name can stand for objects of different types
  - We can change between types by casting

Variables, Operations, Types

- Pattern 1:
  - Processing user input
    - The input function returns a string
    - We often need to cast this string into another type

```
km = input("Enter distance in kilometers: ")
km = float(km)
print("The distance in miles is {:4.3f} miles".format(km*0.621371))
```

- In this sample, the type of km switches from string to float.

Variables, Operations, Types

- The meaning of operands can depend on the type
- Example: `print(a*b)`
    - If both a and b are numerical types (int or float) then the asterisk is a multiplication
    - If one is a string and the other an integer, then the asterisk stands for a replication operation

Flow Control

- Python uses if-statements and for- and while-loops for flow control
- Python blocks are defined by indentation
    - Indentation needs to be consistent within a Python module

Flow Control

- Indentation matters a lot

```
choice = int(input("Enter a number between 1 and 3 "))
if choice != 1:
    if choice == 2:
        print("Well chosen")
    else:
        print("Horrible choice")
```

```
choice = int(input("Enter a number between 1 and 3 "))
if choice != 1:
    if choice == 2:
        print("Well chosen")
else:
    print("Horrible choice")
```

- The placement of the else decides to which if it belongs!

Flow Control

- The range-function creates an iterator
  - There is no problem having a horribly large range because no memory is wasted
  - The first argument is the start (no problem here)
    - Default is 0
  - The second and sometimes only argument is the **stop** value, **not the last value**
  - The third argument is the stride, this can be negative

Flow Control

- Almost any expression can be automatically converted into a boolean value
  - This is perfectly legitimate

```
def successor(n):
    if n%2:
        return 3*n+1
    else:
        return n//2
```

if n is odd, n%2 is 1, and therefore True

  - For odd *n* we multiply by three and add one
  - For even *n* we divide by 2 (as integers)

Flow Control

- If you want to test whether a list is not empty, the **Pythonesque** way is to say:

```
if lista:
    print(“list is not empty”)
```

Flow Control

- While loops:
  - While the condition is true, execute the while-block
  - A frequent error is to forget to update the condition

```
def sum(n):
    counter = 1
    accu = 0
    while counter < 10:
        accu + 1/(counter**2+counter + 1)
    return accu
```

- This while loop never terminates because the condition is always true and never changes

Flow Control

- Breaking out of a while loop:
  - We can break out of the while loop by using the command **`break`**
  - We can break out of the current execution of the while loop, and continue with the next execution of the while loop using the command **`continue`**

Flow Control

- Pattern:
  - Calculating sums and products
  - Use an accumulator (the sum or product) properly initialized
    - 0 for summing, 1 for multiplying
  - Add inside a for loop

Flow Control

- **Calculating** $\sum_{i=1}^{100} i^4 = 1 + 16 + \ldots + 100000000$

```
def sum():
    accu = 0
    for i in range(1, 101):
        accu += i**4
    return accu
```

Initializing the accu to zero, because we are summing up

This is the **stop** value, the sum only goes to 100

Accumulating into the accumulator

Returning the value **outside** of the for loop

Functions

- Python functions are a vast subject.
  - Up till now, we only see a small part of it.
  - Functions are defined using the def command, followed by the function name, and a pair of parameters that encloses a potentially empty list of parameters.
  - Later, we will learn how to
    - Define anonymous functions
    - Use named arguments
    - Use variable number of arguments
    - Expand lists and dictionaries into arguments

Functions

- Pattern:
  - Transformations
    - Change measurements

```
def km2m(km):
    return km*0.621371
```

Function Name

List of parameters

Return value calculated in return statement