Functions

• Collection of instructions that perform a task as:
  o Printing your name and course.
  o Calculating the average of set of numbers.
  o Editing a picture or video.
Functions

• Like in algebra, a function is a kind of “box” into which you put one value and out comes another. We represent (“denote”) a function by a name (in math we use f or F).
Why to use a function?

• If we define a function to perform a task, then we will write it once then we can use it (or call it) many times.
How to write functions?

```python
def functionName():
    statement #1
    statement #2
    ...
```
• def SayHello():
    print(“Hello world!”)
    print(“--From Python”)

Note: 1. Never forget the colon(:)
    2. Align the statements in one function
• No input arguments, no variable returned

• def Greet():
    print("Hello Jack")
    print("Hello Tom")
    def main():
        Greet()
• Multiple arguments, returns one result

• def Average(a, b):
  return (a+b)/2

def main():
  ave = Average(3, 4)
• Returns more than one variable

• def getabc():
    a = "Hello"
    b = "World"
    c = "!"
    return a,b,c

def main():
    a, b, c = getabc()
• def Sum(a, b, c):
  return (a+b+c)
def Greet(name, GPA):
  print("Hello", name)
  print("You have a GPA of ", GPA)
def Div(a, b):
  return a/b
def Mul(a, b):
  return a*b

main()
• # assign values to variables
  
  G1 = 4
  G2 = 3.7
  G3 = 3.7
  C1 = 3
  C2 = 1
  C3 = 2
  name = "Tom"
• GPA = Div(Sum(Mul(G1, C1), Mul(G2, C2), Mul(G3, C3)), Sum(C1, C2, C3))
  Greet(name, GPA)
  input()

• The output of Mul() is input of Sum()
• The output of Sum() is the input of Div()
Booleans

- Boolean (logical) expressions:
  - An expression that can assume only the true or false value
  - We use logical expression in everyday language:
    - *If today is raining, then bring an umbrella when you go out.*
    - *today is raining* is a logical expression: its value can be either true or false.
  - Other examples:
    - assume $x = 4$
    - $x > 3$
    - *(true)*
    - assume $str = \text{"abc"}$
    - $type(str) == \text{int}$
    - *(false)*
Booleans

- Boolean operators:
  - And, or, not

<table>
<thead>
<tr>
<th>a and b</th>
<th>P</th>
<th>Q</th>
<th>P and Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>a and true</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>x &gt; 0 and x &lt;= 2</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>y &gt; 0 and y &gt;= 3 (overlapped)</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F and F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Booleans

- Boolean operators:
  - And, or, not

  \[
  \begin{align*}
  a & \text{ or } b \\
  a & \text{ or } \text{true} \\
  x \leq 0 & \text{ or } x > 2 \\
  x > 5 & \text{ or } x < 10 \text{ (always true)}
  \end{align*}
  \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
• Boolean operators:
  – And, or, not

  not a
  not (not a)
  not x > 3
  DeMorgan’s law
  not (a or b) == (not a) and (not b)
  not (a and b) == (not a) or (not b)

<table>
<thead>
<tr>
<th>P</th>
<th>not P</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Booleans

- \((P \text{ and } (\neg Q)) \text{ or } ((\neg P) \text{ and } Q)\)
  - It has a name: XOR
  - Can you do this?

\[
\begin{array}{ccc}
P & Q & P \text{ xor } Q \\
T & T & \text{T} \\
T & F & \text{F} \\
F & T & \text{T} \\
F & F & \text{F} \\
\end{array}
\]
Does the result look familiar?
How about this: Let T=1, F=0

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Q</td>
<td>P xor Q</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Yes, it is the sum of binary numbers
Decision Structures

• **If** statement
  
  – An if statement takes a logical expression and evaluates it.
  
  – If it is true, the statements in if block are executed, otherwise, they are not executed.

**Simple decision**

```
x = 5
if x>1:
    print("print something")
```

*The string is printed*

```
x = 0
if x>1:
    print("print something")
```

*Does nothing!*
Decision Structures

• Two-way decision

```python
a = 45
if a < 100:
    print("a is small")
else:
    print("a is large")

>>> a is small
```

```python
a = 153
if a < 100:
    print("a is small")
else:
    print("a is large")

>>> a is large
```
Decision Structures

- Two-way decision

\[ a < 100? \]

- no
  - “a is large”

- yes
  - “a is small”
Decision Structures

• Multi-way decision

```python
a = 1.5
if a > 2:
    print("a>2")
else:
    if a > 1:
        print("1<a<=2")
    else:
        print("a<=1")

>>> 1<a<=2
```

```python
a = 1.5
if a > 2:
    print("a>2")
elif a > 1:
    print("1<a<=2")
else:
    print("a<=1")

>>> 1<a<=2
```
Decision Structures

- Multi-way decision
For vs While Loop

- “How long do I have to shower?” You have two replies you could make:
  - a) 10 minutes
  - b) Until you are clean

- a) When programming, the first answer would be portrayed as a for-loop because we focus on exactly how long the shower will continue:
  
  ```python
  for minutes in range (0,9):
      shower
  ```

- b) The second answer would be portrayed as a while-loop because the length of the shower is undetermined; we instead focus on the condition of cleanliness:
  
  ```python
  while you are not clean:
      shower
  ```
For vs While Loop

```python
for count in range(0,9):
    print("The count is:", count)
print("for loop ended")

count = 0
while count < 9:
    print("The count is:", count)
    count = count + 1
print("while loop ended")
```
Rules of **While** Loops

- Command name: **while**

- Boolean condition *(in the previous example: count < 9)*
  - determines the termination of loop

- A colon ("::")

- And a *block* *(the indented lines of code)*
Interactive loop

• Using while loop, we can write interactive loops

```python
count = 0
str = "Yes"
while str == "Yes":
    print("The count is:",count)
    count = count + 1
    str = input("continue? Yes or No:")
print("while loop ended")
```
• Question?