Functions with Default Arguments and Anonymous Functions

Python
Marquette University
Functions with Default Arguments

• We have created functions that have *positional* arguments

• Example:

```python
def fun(foo, bar):
    print(2*foo+bar)

fun(2, 3)
```

• When we invoke this function, the first argument (2) gets plugged into variable foo and the second argument (3) get plugged into variable bar
Keyword (Named) Arguments

• We can also use the names of the variables in the function definition.

• Example: (we soon learn how to deal better with errors)

```python
def quadratic(a, b, c):
    if b**2-4*a*c >= 0:
        return -b/(2*a) + math.sqrt(b**2-4*a*c)/(2*a)
    else:
        print("Error: no solution")

print(quadratic(1, -4, 4))  #CALL BY POSITION
print(quadratic(c=4, a=1, b=-4))  #CALL BY KEYWORD
```
Keyword (Named) Arguments

- Keyword arguments have advantages
  - If you have a function with many positional arguments, then you need to carefully match them up
  - At least, you can use the help function in order to figure out what each argument does, if you named them well in the function definition

```python
>>> help(quadratic)
Help on function quadratic in module __main__:
quadratic(a, b, c)
```
Keyword (Named) Arguments

• You can force the user of a function to use keywords by introducing an asterisk into the definition of the function:
  
  • All arguments after the asterisk need to be passed by keyword
  
  • The arguments before the asterisk can be positional

```python
def function ( posarg1, *, keywarg1 ):
  def fun(a, b, *, c):
    ...
  print(fun(2, 3, c=5))
```
Default arguments

• You have already interacted with built-in functions that use default arguments
  • Print:
    • end: How the string is terminated (default is new-line character)
    • sep: What comes between different outputs (default is space)
    • file: Location of output (default is “standard output”)

```python
>>> for i in range(10):
    print(i**3, end=', ')

0, 1, 8, 27, 64, 125, 216, 343, 512, 729,
```
Default arguments

- Defining default arguments is easy
  - Just use the arguments with default arguments last and assign default values in the function definition

```python
def fun(a, b, c=0, d=0):
    return a+c*b+d*a*b

print("10+0*1=", fun(10,1), sep="")
print("10+5*1=",fun(10,1,c=5), sep="")
print("10+0*1+3*10*1=", fun(10,1,d=3), sep="")
print("10+5*1+5*10*1=", fun(10,1,c=5,d=5), sep="")
```

```
10+0*1=10
10+5*1=15
10+0*1+3*10*1=40
10+5*1+5*10*1=65
```
Default Arguments

How to write readable code:

- Named arguments and default arguments with well-chosen names make code more readable
- Most effort in software engineering goes towards maintaining code
Anonymous Functions

- Up till now, we used the def-construct in order to define functions
- Sometimes it is necessary to pass functions to another function, but not necessary to define the argument for future uses
Anonymous Function

- Example:
  - Numerical Differentiation
    - Derivative of a function $f$ at a point is the slope of the tangent
    - Approximated by a secant

Slope of the secant is \( \frac{f(x + \delta) - f(x - \delta)}{2\delta} \)
Anonymous Functions

- The slope of the secant is the difference of values over the difference of arguments:
  \[
  \frac{f(x + \delta) - f(x - \delta)}{2\delta} = \frac{f(x + \delta) - f(x - \delta)}{x + \delta - (x - \delta)}
  \]

- If \( \delta \) is small, then this is a good approximation of the derivative
Anonymous Functions

- A simple method for derivation uses a fixed, but small value for $\delta$.

```python
def derivative(function, x):
    delta = 0.000001
    return (function(x+delta)-function(x-delta))/(2*delta)
```

- To test this, we try it out with sine, whose derivative is cosine

```python
for i in range(20):
    x = i/20
    print(x, math.cos(x), derivative(math.sin, x))
```
Anonymous Functions

* It turns out that the numerical derivative is quite close in this test

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
<th>f(x+dx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1.0</td>
<td>0.9999999999998334</td>
</tr>
<tr>
<td>0.05</td>
<td>0.9987502603949663</td>
<td>0.9987502603940601</td>
</tr>
<tr>
<td>0.1</td>
<td>0.9950041652780257</td>
<td>0.9950041652759256</td>
</tr>
<tr>
<td>0.15</td>
<td>0.9887710779360422</td>
<td>0.9887710779310499</td>
</tr>
<tr>
<td>0.2</td>
<td>0.980665778412416</td>
<td>0.980665778519901</td>
</tr>
<tr>
<td>0.25</td>
<td>0.9689124217106447</td>
<td>0.9689124216977207</td>
</tr>
<tr>
<td>0.3</td>
<td>0.955336489125606</td>
<td>0.9553364891112803</td>
</tr>
<tr>
<td>0.35</td>
<td>0.9393727128473789</td>
<td>0.9393727128381713</td>
</tr>
<tr>
<td>0.4</td>
<td>0.9210609940028851</td>
<td>0.9210609939747094</td>
</tr>
<tr>
<td>0.45</td>
<td>0.9004471023526769</td>
<td>0.9004471023255078</td>
</tr>
<tr>
<td>0.5</td>
<td>0.8775825618903728</td>
<td>0.8775825618978494</td>
</tr>
<tr>
<td>0.55</td>
<td>0.8525245220595057</td>
<td>0.8525245220880606</td>
</tr>
<tr>
<td>0.6</td>
<td>0.8253356149096783</td>
<td>0.8253356149623414</td>
</tr>
<tr>
<td>0.65</td>
<td>0.7960837985490559</td>
<td>0.7960837985487856</td>
</tr>
<tr>
<td>0.7</td>
<td>0.7648421872844885</td>
<td>0.7648421873063249</td>
</tr>
<tr>
<td>0.75</td>
<td>0.7316888688738209</td>
<td>0.7316888688824186</td>
</tr>
<tr>
<td>0.8</td>
<td>0.6967067093471655</td>
<td>0.6967067094354462</td>
</tr>
<tr>
<td>0.85</td>
<td>0.6598314588498222</td>
<td>0.659831459119798</td>
</tr>
<tr>
<td>0.9</td>
<td>0.6216099682706645</td>
<td>0.6216099682765375</td>
</tr>
<tr>
<td>0.95</td>
<td>0.5816830894638836</td>
<td>0.5816830894733727</td>
</tr>
</tbody>
</table>
Anonymous Functions

• Notice that in the test, we specified math.sin and not math.sin(x),
• The former is a function (which we want)
• The latter is a value (which we do not want)

```python
for i in range(20):
    x = i/20
    print(x, math.cos(x), derivative(math.sin, x))
```
Anonymous Functions

• To specify a function argument, I can use a **lambda-expression**
  • Lambda-expressions were used in Mathematical Logic to investigate the potential of formal calculations

\[
\text{\texttt{lambda x : 5*x**x-4*x+3}}
\]

• Lambda expression consists of a keyword lambda
  • followed by one or more variables
  • followed by a colon
  • followed by an expression for the function
• This example implements the function \( x \to 5x^2 - 4x + 3 \)
Anonymous Functions

- To test our numerical differentiation function, we pass it the function $x \rightarrow x^2$, which has derivative $2x$

```python
for i in range(20):
    x = i/20
    print("{:5.3f} {:5.3f} {:5.3f}".format(
        x,
        derivative(lambda x: x*x, x),
        2*x))
```
Anonymous Functions

- Since we are rounding to only three digits after the decimal point, we get perfect results:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.050</td>
<td>0.100</td>
<td>0.100</td>
</tr>
<tr>
<td>0.100</td>
<td>0.200</td>
<td>0.200</td>
</tr>
<tr>
<td>0.150</td>
<td>0.300</td>
<td>0.300</td>
</tr>
<tr>
<td>0.200</td>
<td>0.400</td>
<td>0.400</td>
</tr>
<tr>
<td>0.250</td>
<td>0.500</td>
<td>0.500</td>
</tr>
<tr>
<td>0.300</td>
<td>0.600</td>
<td>0.600</td>
</tr>
<tr>
<td>0.350</td>
<td>0.700</td>
<td>0.700</td>
</tr>
<tr>
<td>0.400</td>
<td>0.800</td>
<td>0.800</td>
</tr>
<tr>
<td>0.450</td>
<td>0.900</td>
<td>0.900</td>
</tr>
<tr>
<td>0.500</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>0.550</td>
<td>1.100</td>
<td>1.100</td>
</tr>
<tr>
<td>0.600</td>
<td>1.200</td>
<td>1.200</td>
</tr>
<tr>
<td>0.650</td>
<td>1.300</td>
<td>1.300</td>
</tr>
<tr>
<td>0.700</td>
<td>1.400</td>
<td>1.400</td>
</tr>
<tr>
<td>0.750</td>
<td>1.500</td>
<td>1.500</td>
</tr>
<tr>
<td>0.800</td>
<td>1.600</td>
<td>1.600</td>
</tr>
<tr>
<td>0.850</td>
<td>1.700</td>
<td>1.700</td>
</tr>
<tr>
<td>0.900</td>
<td>1.800</td>
<td>1.800</td>
</tr>
<tr>
<td>0.950</td>
<td>1.900</td>
<td>1.900</td>
</tr>
</tbody>
</table>
Anonymous Functions

• I can even use lambda expressions as an alternative way of defining functions:

\[
\text{norm} = \text{lambda } x, y: \text{math.sqrt}(x*x+y*y)
\]

• Since there are two variables, norm is a function of two arguments:

\[
\text{print}(\text{norm}(2.3, 1.7))
\]