Default Values

- Parameters of a function can have default values
  - Assigned to them when no values are passed to them
- Format for definition using default values

```
def functionName(par1, par2, par3=value3, par4=value4):
```

Default Values

```
def total(w, x, y=10, z=20):
    return (w ** x) + y + z
```

Function Call	Value	Calculated As
`total(2, 3)`	38	$2^3 + 10 + 20$
`total(2, 3, 4)`	32	$2^3 + 4 + 20$
`total(2, 3, 4, 5)`	17	$2^3 + 4 + 5$

Default Values

- Example: Program gives user three tries to answer question

```
def main():
    ## A quiz
    q = "What is the capital of California? "
    a = "Sacramento"
    askQuestion(q, a)

def askQuestion(question, answer, numberOfTries=3):
    numTries = 0
    while numTries < numberOfTries:
        numTries += 1
        ans = input(question)
        if ans == answer:
            print("Correct!")
            break
    if ans != answer:
        print("You have used up your allotment of guesses.")
        print("The correct answer is", answer + '.')
```

Passing by Parameter Name

- Arguments can be passed to functions by using names of the corresponding parameters
  - Instead of relying on position
- Given

```
def total(w, x, y=10, z=20):
    return (w ** x) + y + z
```

  - Could use

```
total(w=2, x=3) or total(x=3, w=2)
```

Passing by Parameter Name

- Example: Several ways to pass values

```
def main():
    ## Demonstrate the passing of values.
    print("Balance:")
    print("${0:,.2f}".format(balance(1000, 5)))
    print("${0:,.2f}".format(balance(1000, 5, .04)))
    print("${0:,.2f}".format(balance(1000, intRate=.04, numYears=5)))
    print("${0:,.2f}".format(balance(numYears=5, prin=1000)))
    print()
    print("${0:,.2f}".format(balance(1000, 5, .03)))
    print("${0:,.2f}".format(balance(1000, intRate=.03, numYears=5)))
    print("${0:,.2f}".format(balance(intRate=.03, numYears=5, prin=1000)))
    print("${0:,.2f}".format(balance(numYears=5, intRate=.03, prin=1000)))

def balance(prin, numYears, intRate=.04):
    return prin * ((1 + intRate) ** numYears)
```

```
Balance:
$1,216.65
$1,216.65
$1,216.65
$1,216.65

$1,159.27
$1,159.27
$1,159.27
$1,159.27
```

Custom Sorting

- Functions can be used to order the items by any criteria we choose
- With a list of strings, we can sort them by
  - Length
  - Last characters
  - Number of vowels
  - … by many other properties

Custom Sorting

- Example: Program sorts list of words using each of three properties mentioned above

```
def main():
    ## Custom sort a list of words.
    list1 = ["democratic", "sequoia", "equals", "brrr", "break", "two"]
    list1.sort(key=len)
    print("Sorted by length in ascending order:")
    print(list1, '\n')
    list1.sort(key=lastCharacter)
    print("Sorted by last character in ascending order:")
    print(list1, '\n')
    list1.sort(key=numberOfVowels, reverse=True)
    print("Sorted by number of vowels in descending order:")
    print(list1)
```

```
def lastCharacter(word):
    return word[-1]
```

```
def numberOfVowels(word):
    vowels = ('a', 'e', 'i', 'o', 'u')
    total = 0
    for vowel in vowels:
        total += word.count(vowel)
    return total

main()
```

```
Sorted by length in ascending order:
['two', 'brrr', 'break', 'equals', 'sequoia', 'democratic']

Sorted by last character in ascending order:
['sequoia', 'democratic', 'break', 'two', 'brrr', 'equals']

Sorted by number of vowels in descending order:
['sequoia', 'democratic', 'equals', 'break', 'two', 'brrr']
```

The *sorted* Function

- Contrast with *sort* function
  - *sort* function alters order of items in a list
  - *sorted* function returns a new ordered list

```
list2 = sorted(list1)
```

  - Both can make use of the optional arguments *key* and *reverse*.
  - *sorted* function also can be used with lists, strings, and tuples

The *sorted* Function

list1 = ["white", "blue", "red"]

Statement	Output of print(list2)
`list2 = sorted(list1)`	`['blue', 'red', 'white']`
`list2 = sorted(list1, reverse=True)`	`['white', 'red', 'blue']`
`list2 = sorted(list1, key=len)`	`['red', 'blue', 'white']`
`list2 = sorted("spam")`	`['a', 'm', 'p', 's']`

List Comprehension

- Simpler way to apply a certain function to each item of a list
  - Use list comprehension

```
list2 = [f(x) for x in list1]
```

- The *for* clause in a list comprehension can optionally be followed by an *if* clause.

```
[g(x) for x in list1 if x % 2 == 1]
```

List Comprehension	Result
`[ord(x) for x in "abc"]`	`[97, 98, 99]`
`[x ** 2 for x in range(3)]`	`[0, 1, 4]`

Lambda Functions or Expressions

- One-line mini-functions
  - Can be used where a simple function is required.
  - Compute a single expression
  - Cannot be used as a replacement for complex functions
- Format

```
lambda par1, par2, ...: expression
```

  - Where *expression* is the value to be returned

Lambda Functions

```
add_numbers_and_five = lambda number1, number2: number1 + number2 + 5

print(add_numbers_and_five(number1=4, number2=3))
```

- Rather than using **def**, the word **lambda** is used
- No brackets are required, but any words following the **lambda** keyword are created as parameters
- The colon is used to separate the parameters and the expression. In this case, the expression is:

  number1 + number2 + 5.
- There's no need to use the **return** keyword—the lambda does this for you automatically.

Lambda Functions

- Example: Function sorts names by their surnames

```
names = ["Dennis Ritchie", "Alan Kay", "John Backus", "James Gosling"]
names.sort(key=lambda name: name.split()[-1])
nameString = ", ".join(names)
print(nameString)
```

[Run]

```
John Backus, James Gosling, Alan Kay, Dennis Ritchie
```

  - Lambda function highlighted

Lambda Functions

- Examples

```
>>> def cube(x): return x**3

>>> print(cube(9))
729
>>> result = lambda x: x**3
>>>
>>> print(result(9))
729
```

```
>>> sample_list = [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
>>>
>>> print(list(filter(lambda x: x % 9 == 0, sample_list)))
[9, 18, 27]
>>>
```

Filter function in Python (https://www.w3schools.com/python/ref_func_filter.asp)