Computer Science Principles

CHAPTER 2 – COMPUTER PROGRAMMING FUNDAMENTALS

1

Announcements

Reading: Read Chapter 2 in the Conery textbook (Explorations in Computing)

- Be sure to do the "Tutorial" sections

Python programs from slides posted

Acknowledgement: These slides are revised versions of slides prepared by Prof. Arthur Lee, Tony Mione, and Pravin Pawar for earlier CSE 101 classes. Some slides are based on Prof. Kevin McDonald at SBU CSE 101 lecture notes and the textbook by John Conery.

2

Expressions

An **Expression** represents something like a number, string or value

"Hello, world!" is an expression
- It has a value
- In this case, it's a **string** (a sequence of characters)

Numbers are also expressions
- **5** is an **integer** expression
  - Recall that an integer is zero, or a positive or negative whole number with no fractional part
- **12.36** is a **floating-point** expression
  - **Floating-point** is a format that computers use to represent **real** numbers
  - Recall that a real number is zero, or a positive or negative number that might have a fractional part

3

Expressions

An expression may also look like **2 + 9**
- This represents an addition
- Some of the simplest expressions in Python involve these arithmetic expressions

Each number is called an **operand** – these are the inputs to the function

The plus (+) is an **operator** – this is telling what to do with the input (in this case, add)

The expression may consist of many different operators and operands.

4

Arithmetic in Python

The symbols used for operators are commonly used in other languages and applications (e.g., spreadsheets)

- add: +
- subtract: -
- multiplication: *
- division: /
- integer division: // (divides and rounds down to the nearest whole number)
- remainder: % (gives the remainder of an integer division, also called "**modulo**")
- exponentiation: **

5

Examples of arithmetic in Python

11 + 5 ➔ 16

11 – 5 ➔ 6

11 * 5 ➔ 55

11 / 5 ➔ 2.2

11 // 5 ➔ 2

- This example shows **integer division**. Any remainder is discarded.

11 % 5 ➔ 1

- The computer divides 11 by 5 and returns the remainder (which is 1), not the quotient (which is 2).
- The % is performing the "**modulo operation**"

6

Examples of arithmetic in Python

What happens when you have:

2 * (3 + 2) – 4 / 2

➔ **8**

Arithmetic in Python follows the PEMDAS rule:

1. First, evaluate all expressions in parentheses (P)
2. Then, perform exponentiations (E)
3. Next, perform multiplications (M) and divisions (D) in left-to-right order
4. Finally, perform additions (A) and subtractions (S) in left-to-right order

7

Arithmetic in Python

The ** operator does exponentiation or raises a number to a power

For example: **2 ** 5** ➔ 32 because $2^5 = 32$

Recall raising a number to the power of ½ is the same as taking a square root

- So 16 ** 0.5 would be the same as √16 which is 4

8

Expressions

Python also has **Boolean** expressions, which are expressions that can be **True** or **False**

So there are at least three kinds of data in Python programming:

- Strings
- Numbers
- true/false (Boolean) values

In computer programming, there are a wide variety of types of data because there are a wide variety of problems that computers can help solve

Variables

A **variable** in computer programming is similar to the concept of a variable in mathematics

- A name for some value or quantity of interest in a given problem

```
billTotal = 10.50
```

The variable **named** billTotal is **assigned** the value 10.50

Now if we later refer to the billTotal, it will be 10.50

In a program, variables can store a person's age, GPA, name, or almost any other kind of information

- The value is temporarily stored in the main memory of the computer while the program is running
- A variable is a kind of **identifier** because it identifies (names) something in source code

Assignment statements

When we give a value to a variable, we are writing an **assignment statement**

An assignment statement consists of a variable name, the equals sign, and a value or expression

Examples:

- **length = 3.5**
- **total = 2 + 5** (total becomes 7)
- **firstName = "Susan"**

These examples show three different data types: a real number, an integer, and a string

11

Assignment statements

After assigning a value to a variable, you can change the value of the variable with another assignment statement:

```
total = 5
... other code here ...
total = 17 + 6
... more code ...
```

Variables can also appear on the right-hand side (RHS) of an assignment statement:

```
next_year = this_year + 1
total_earnings = income - taxes
```


12

Variables

It is important to choose variable names that are informative and helpful

Do you have any idea what these variables represent?
tb = st + tx + tp

Do these names help?
total_bill = subtotal + tax + tip

Note how the underscore is used to separate words that define the identifier
- Spaces are not allowed in variable names

13

Variables

A Python variable name may contain lowercase letters, uppercase letters, digits and underscores
- First character must be a letter or underscore

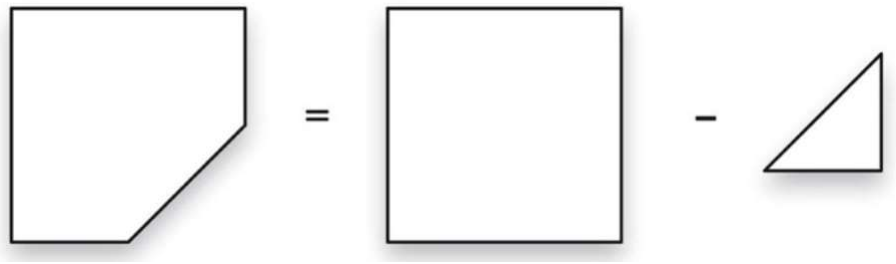
Lowercase and uppercase letters are treated as completely different characters
- Because of this we say that Python is a **case-sensitive language**
- **First_Name**, **first_name** and **FIRST_NAME** would all be treated as different identifiers

There are a number of **keywords** built into the Python language that have pre-defined meanings
- Predefined keywords may not be used as variables
- Examples of predefined keywords are: "**if**" "**for**" "**and**"

14

Example: Area calculation

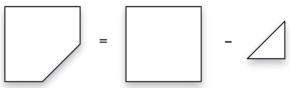
Want to compute the area of a square countertop with one corner cut off, as shown here



15

Example: Area calculation

Assume that the triangular cut-out begins halfway along each edge



If it is 100 cm long on each side, we can write a statement like this:

- **area = 100**2 - 50*50/2**

Note that this code has a few issues:

- It's just a formula with no explanation of what the numbers mean
- The code works only for countertops exactly 100 cm long. What if we had countertops of other sizes?

16

Example: Area calculation

Consider the first issue: lack of clarity

```
# area = area of square - area of triangle
# area of triangle is 1/2 base*height
area = 100**2 - 50*50/2
```

The lines beginning with the # symbol are called **comments**

- Comments are notes that the programmer writes to explain what the program does
- Comments do not affect the input or output of the program or anything about how it runs

17

Example: Area calculation

Now let's address the other issue: lack of generality

```
side = 100
square = side**2
triangle = (side / 2)**2 / 2
area = square – triangle
```

To compute the area for a countertop of a different size, simply change the first line:

```
side = 100
```

This code is also more readable; comments aren't needed

- This is an example of **self-documenting code**

The spacing in between variables, numbers, and operator is optional, but is included here to make the formulas easier to read

18

Aside: input statements

To improve the code further, we can make it interactive so that the user can provide the value for the **side**

Do this by writing an **input** statement – An input statement reads a **string** from the keyboard

As part of an input statement, the programmer must give a **prompt** message that tells the user what they should enter

Example: **name = input("What is your name? ")**

The person's name will be assigned to the name variable

- You could also say that we are saving the person's name in the name variable

19

Example: Area calculation

In the case of the area calculation, the user should enter a number, not a string

To collect a floating-point number, use:

- **side = float(input("Enter side length: "))**

If we wanted to only allow integer numbers as input, we would use:

- **side = int(input("Enter side length: "))**

The type chosen – **int** vs. **float** – depends on the application

For this program, read in a float so the user could enter a fraction of a centimeter if desired

The last step is how to display the final result on the computer screen

20

Aside: print statements

print is a Python command

It tells Python to display some text on the screen
- All Python commands are lowercase

The syntax to print a basic message is just this:
- **print("Hello, world!")**

Any text printed with additional print commands will appear on a new line

For Python to print the next output on the same line, do this instead:
- **print("Hello, world!", end="")**
- This means print this message, but do not automatically go to the next line

21

Aside: print statements

To print a number, it must first be converted into a string, like so:
- **print("The area is " + str(area))**
- The assumption here is that **area** is a variable that contains the value we want to print

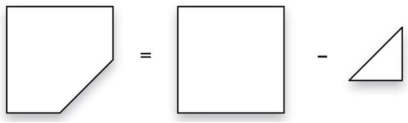
When used in this fashion, the + symbol performs **string concatenation**
- This simply means Python will join the two strings together into one

22

Example: countertop.py

```
# This program prints the area of a
# countertop formed by cutting the
# corner off a square piece of material


side = float(input("Enter side length: "))
square = side**2
triangle = (side/2)**2 / 2
area = square - triangle
print("The area is " + str(area))
```



Example: coins.py

Here is an example that uses the modulo (remainder) operator with integer division:

Given a total number of cents, the computer should print how many dimes, nickels, and pennies are needed to make that change while minimizing the number of coins

- Note: 1 dime = 10 cents, 1 nickel = 5 cents, 1 penny = 1 cent.

Note:

The code will use several variables

It will also need the **str** command to print variables containing numbers to the screen

- Recall that **str** converts a number to a string so that it can be concatenated with other strings

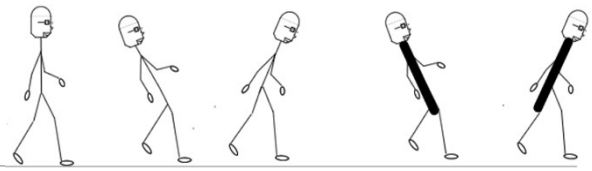
Example: coins.py

```
cents = int(input("Enter the number of cents: "))

dimes = cents // 10
cents = cents % 10
nickels = cents // 5
cents = cents % 5
pennies = cents

print("That number of cents is equal to " +
      str(dimes) + " dimes, " + str(nickels) +
      " nickels and " + str(pennies) + " pennies.")
```


Escape sequences



Escape sequences in programming languages like Python allow printing characters (symbols) on the screen that perform special functions

In Python, some of the escape sequences are:

- **\t** shifts the text to the right by one tab stop
- **\n** prints a newline
- **\"** prints a double quotation mark
- **\'** prints a single quotation mark

A lone backslash character is called the line-continuation character (it's not really an escape sequence, though)

- This symbol is a signal to Python that the current statement spans two or more lines of a file

Example: limerick.py

Source code:

```
print('There was an old man with a beard\n\
Who said, \"It\'s just how I feared!\"\n\
\tTwo owls and a hen\n\
\tFour larks and a wren\n\
Have all built their nests in my beard.')
```

Output:

```
There was an old man with a beard
Who said, "It's just how I feared!"
        Two owls and a hen
        Four larks and a wren
Have all built their nests in my beard.
```


Additional Arithmetic in Python

The constant π is built into Python

First the programmer must make it available by importing the **math** module:

- **import math**

Then **math.pi** can be used in expressions

- **math.pi * 2**

A Python module is a file consisting of Python source code that are all related somehow

Functions

Python's math module contains code related to mathematical functions
- The library has numbers (e, π , etc.)
- Also has a variety of useful mathematical functions (e.g. calculate the cosine of a variable)

In *programming*, a **function** is a name given to a set of statements that perform a well-defined task

For example, the **input** function performs a task (getting user input) and also returns the value entered by the user

```
name = input("What is your name? ")
```

print, **int**, **float**, and **str** are also functions

The next example introduces a new function, **format**, that lets the programmer format numerical output

Example: BMI calculator

Once numbers are stored in variables, they can be used in calculations

The Body Mass Index (BMI) is a metric used to gauge a person's general health

Given a person's weight in pounds and total height in inches, a person's BMI is calculated as:
- BMI = (weight * 703) / height^2

A BMI in the range of 18.5-24.9 is considered “healthy”

We want to create a program that calculates and prints a person's BMI based on entered values.

However, we want to ensure the BMI is printed with 3 digit decimal precision (e.g. 19.421)
- By default it will print 15 decimal digit, e.g. 19.421004314691235

Example: BMI calculator

To print a number to a designed number of digits, use the **format** function

Suppose there is a variable **total** to be printed with two decimal places. Here is how to do it:

```
print("Total: " + "{:.2f}".format(total))
```

If we wanted four digits, we would write **{:.4f}** instead

Note that when using the format method, do not also use **str** to print a number

31

Example: bmi_v1.py

```
weight = float(input('Enter weight in pounds: '))
feet = float(input('Enter feet portion of height: '))
inches = float(input('Enter inches portion of height: '))

total_inches = feet * 12 + inches

bmi = (weight * 703) / total_inches ** 2

print('Your BMI is ' + str(bmi))
print('Your BMI is ' + '{:.3f}'.format(bmi))
```

The first print statement gives the BMI with full accuracy. The second print statement rounds to 3 decimal places.

The blank lines are present to make the code more readable. They do not affect program execution in any way.

More about format function: https://www.python-course.eu/python3_formatted_output.php

32

Other functions in Python

Some examples:

```
type(45)
int(34.56)
float(45)
str(3421)
len('apple')
round(2.32)
abs(-45)
pow(2, 3)
help(pow)
...
```

```
import math
math
math.log(10)
math.log10(10)
math.log10(1e6)
radians = 0.7
math.sin(radians)
math.sqrt(3)
...
```

```
import random
random.random()
random.randint(0,100)
```

Try these on a Python Console or as part of a program

Function composition

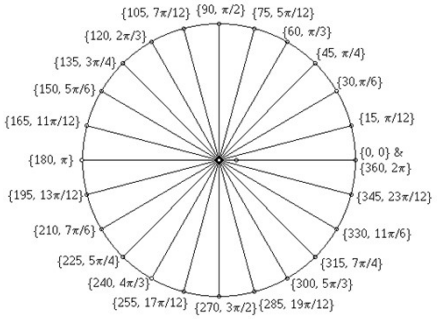
Can compose functions as is done in mathematics, e.g., f(g(x,y))

```
import math
radians = 0.7
print(math.degrees(radians))

math.radians(math.degrees(radians))

radians = 0.3
math.acos(math.cos(radians))

pow(abs(-3), round(5.6))
```



Defining new functions

Functions in program have many benefits, including:

- They make code easier to read and understand
  - ➔ don't need to know the details of how or why a function works
- They allow code to be used more than once (code re-use)

To define a new function in Python we use a **def** statement

- Consider writing a function that computes a person's Body Mass Index
- Can then call this function as many times as desired

The alternative would be to copy and paste the code multiple times

- First rule of programming: don't repeat yourself!

35

Creating new functions

From mathematics: a 2 step process

1. Define a function, once
   f(x, y) = x*y + 1
2. Apply/Use/Invoke/Call the function, as many times as desired
   f(2,3) = 2*3 + 1 = 7

Do the same in programming: again a 2 step process

1. Define a function, once

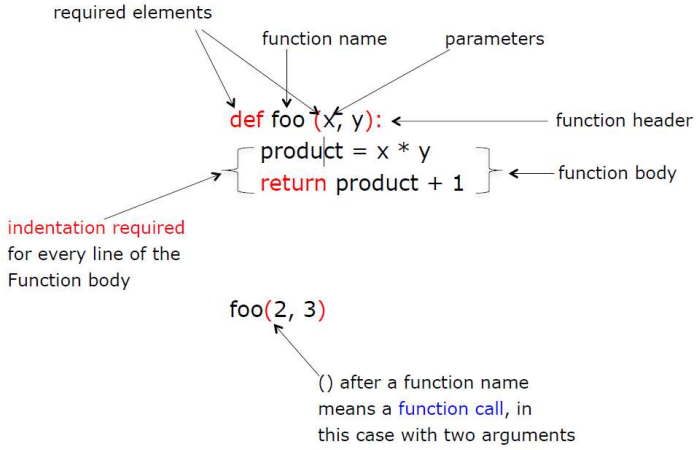
```
def f(x, y):
    return x * y + 1
```

2. Apply/Use/Invoke/Call the function, as many times as desired

```
f(2, 3)
```


36

Mechanics of defining/calling a function



37

Parameters and arguments

Function can have zero or more parameters

- Function may be defined with *formal parameters*

Example:

```
def multAdd(a, b, c):
    return a * b + c

print(multAdd(1, 2, 3))

print(multAdd(2.1, 3.4, 4.3))
print(multAdd(abs(pow(2,3)), 3.2 + 2.3, 45.34))
```

- Functions can be then called with *actual arguments*
- How many? As many as the function needs!

38

Program: flow of execution

```
def message():
    print(1)
    message1()
    print(2)
def message1():
    print('a')
    message2()
    print('b')
def message2():
    print('middle')

message()
```

Note: this is a *program* with three functions and it starts with a call to *message*.

Output:

1

a

middle

b

2

Fruitful functions

square is an example of a *fruitful function*

- It returns a value when it is called

```
// fruitful function
def square(n):
    return n * n

print(square(3))
```

What will get printed when this code runs?

Void functions

announce below is an example of a *void function*

- It does not return any useful value when it is called; it only prints a value

```
// void function
def announce(message):
    print(message)

announce('hello!')
```

What would happen if you tried to print your announce call?

41

Void functions – Part 2

```
// void function
def announce(message):
    print(message)

print(announce('hello!'))
```

Since announce does not have a return statement, it returns **None**

So the statement `print(announce('hello!'))` will print **None**.

42

Example: bmi_v2.py

```
# Function definition
def bmi(w, h):
    return (w * 703) / (h ** 2)

# main is to use the function defined above.
def main():
    weight = float(input('Enter weight in pounds: '))
    feet = float(input('Enter feet portion of height: '))
    inches = float(input('Enter inches portion of height: '))

    total_inches = feet * 12 + inches
    my_bmi = bmi(weight, total_inches)
    print('Your BMI is ' + '{:.3f}'.format(my_bmi))

# This sets up a call to the function main.
main()
```

Note how a program is organized.

Why functions? Abstraction

One of the most important concepts in computer science is **abstraction**

- Give a **name** to a group of statements and use it.
- E.g. bmi(…)

From the outside, the details are hidden

- Only care that calling this function will do a desired computation

Functions thus allow complex problems to be solved by dividing it into smaller, more manageable sub-problems

- This process is called **problem decomposition** (also **functional decomposition**)

Example: bmi_v3.py

Here's an alternative way of implementing the bmi function

- It illustrates proper indentation and relies on **two local variables**, **numerator** and **denominator**

A **local variable** is a variable accessible only inside the function where it is created

45

Example: bmi_v3.py

```
def bmi(w, h):
    numerator = w * 703
    denominator = h ** 2
    return numerator / denominator

def main():
    weight = float(input('Enter weight in pounds: '))
    feet = float(input('Enter feet portion of height: '))
    inches = float(input('Enter inches portion of height: '))

    total_inches = feet * 12 + inches
    my_bmi = bmi(weight, total_inches)
    print('Your BMI is ' + '{:.3f}'.format(my_bmi))

main()
```


46

Example: Distance calculator

Example: A distance traveled is provided in miles, yards, and feet (e.g. 3 miles, 68 yards, 16 feet)

- Need this to be converted to total inches traveled and print the result
- This requires some unit conversions

Recall the following equivalences:

- 1 foot = 12 inches
- 1 yard = 3 feet
- 1 mile = 5,280 feet

Finally, to print a comma every three digits we can use the formatting string '{:,}' when printing an integer

Example: distance.py

```
def distance(m, y, f):
    return (m * 5280 * 12) + (y * 3 * 12) + (f * 12)

def main():
    miles = int(input('Enter the number of miles: '))
    yards = int(input('Enter the number of yards: '))
    feet = int(input('Enter the number of feet: '))

    inches = distance(miles, yards, feet)

    print('Distance in inches: ' + '{:,}'.format(inches))

main()
```

Example: Mortgage calculator

The monthly payment on a fixed-rate mortgage can be calculated using this formula:

- $payment = (r * P)/(1 - (1 + r)^{-n})$

Where:

- P is the principal (the amount we borrowed)
- r is the monthly interest rate as a decimal (i.e., the annual interest rate as a decimal divided by 12)
- n is the number of months the loan will last

Let's also improve formatting the output:

- To include a comma every three digits, write the format string as '{:,.2f}' for floats
- Also, a format string can be saved in a variable if it's needed to format several numbers in the same way:

```
fmt = '{:,.2f}'
```

Now we will create a function to compute the payment

Example: mortgage.py

```
def monthly_payment(borrow_amt, monthly_rate, num_months):
    return (borrow_amt * monthly_rate) / (1 - (1 + monthly_rate) ** -num_months)

def main():
    principal = float(input('Enter principal: '))
    annual_rate = float(input('Enter annual interest rate as a percentage:'))
    years = int(input('Enter term of mortgage in years: '))

    payment = monthly_payment(principal, annual_rate / 12 / 100, years * 12)
    totalPaid = payment * years * 12
    totalInterest = totalPaid - principal

    fmt = '{:,.2f}' # formatter string
    print('Principal: $' + fmt.format(principal))
    print('Annual interest rate: ' + fmt.format(annual_rate) + '%')
    print('Term of loan in years: ' + str(years))
    print('Monthly payment: $' + fmt.format(payment))
    print('Total money paid back: $' + fmt.format(totalPaid))
    print('Total interest paid: $' + fmt.format(totalInterest))

main()
```

Conditional execution

Often an algorithm needs to make a decision

The steps which are executed next depend on the outcome of the decision

Example: a person's income range determines the income taxation rate

- If the income is above a certain minimum, use one tax rate; otherwise, use a lower rate

In Python, an **if statement** allows testing conditions and executing different steps depending on the outcome

51

Example: Tuition calculator

Suppose part-time students (< 12 credits) at a fictional college pay $600 per credit and full-time students pay $5,000 per semester.

Use an if statement to write a short program that implements this logic

52

Example: tuition.py

```
numCredits = int(input('Enter number of credits: '))

if numCredits < 12:
    cost = numCredits*600
    print('A student taking ' + str(numCredits) +
              ' credits is part-time and will pay $' +
              str(cost) + ' in tuition.')
else:
    print('A student taking ' + str(numCredits) +
              ' credits is full-time and will pay
              $5,000 in tuition.')
```


Conditional execution

If statements can also appear in functions:

```
def tax_rate(income):
    if income < 10000:
        return 0.0
   else:
       return 5.0
```

The value returned by the function depends on value passed as an argument to the parameter

Things to note about the **if** statement:

- The words **if** and **else** are keywords
- There is a **colon (:)** at the end of the if and else **clauses**
- The statements to be executed are **indented**

Multi-way if-statements

When an algorithm needs to choose among more than two alternatives, it can use **elif** clauses

- **elif** is short for "else if"

This function distinguishes between three tax brackets:

```
def tax_rate(income):
    if income < 10000:
        return 0.0
    elif income < 20000:
        return 5.0
    else:
        return 7.0
```

Can use as many **elif** parts as needed

55

Boolean expressions

The expressions inside **if** and **elif** statements are special kinds of expressions
The result of these expressions is either **True** or **False**
An expression that evaluates to **True** or **False** is called a **Boolean expression**

Boolean expressions often involve **relational operators**:

- equal to / not equal to
- greater than / greater than or equal to
- less than / less than or equal to

56

Boolean expressions

The notation >= means "greater than or equal to" and is one of six **relational operators** supported by Python:

Mathematical Operator	Python Equivalent	Meaning
=	==	is equal to
≠	!=	is not equal to
>	>	is greater than
≥	>=	is greater than or equal to
<	<	is less than
≤	<=	is less than or equal to

57

Example: Overtime calculator

Someone who works more than 40 hours a week is entitled to "time-and-a-half" overtime pay

How can the following be determined?

1. Whether an employee is entitled to overtime pay
2. If so, how much are they paid?

For #1 use an if statement

For #2 a different calculation is required depending on whether employee will earn overtime pay or not

Regular pay formula: hourly wage × hours worked

The overtime formula has two parts:

- The pay for the first 40 hours
- The pay for additional overtime hours (1.5 x hourly wage)

58

Example: paycheck.py

```
def compute_pay(hours, wage):
    if hours <= 40:
        paycheck = hours * wage
    else:
        paycheck = 40 * wage + (hours - 40) * 1.5 * wage
    return paycheck

def main():
    hours_worked = float(input('Enter # of hours worked: '))
    hourly_wage = float(input('Enter hourly wage: '))

    pay = compute_pay(hours_worked, hourly_wage)
    print('Your pay is $' + '{:.2f}'.format(pay))

main()
```


Example: Hiring decisions

A hiring manager is trying to decide which candidates to hire

Each potential hire is evaluated based on GPA, interview performance, and an aptitude exam

- A GPA of at least 3.3 is worth 1 point
- An interview score of 7 or 8 (out of 10) is worth 1 point; a score of 9 or 10 is worth 2 points
- An aptitude test score above 85 is worth 1 point

Hiring decisions are then based on point totals:

- 0, 1 or 2 total points: Not hired
- 3 total points: hired as a Junior Salesperson
- 4 points: hired as a Manager-in-Training

Example: Hiring decisions

Following is a function that takes these three values and returns the hiring decision as a string

The following Python capabilities will help with this task:

- The **+=** operator can be used to increment a variable by some amount

```
amount += 5
amount = amount + 5 // Exact same as the line above
```

- -=, *= and /= also exist and perform similar operations
- A variable can be used to maintain a running total
- An if-statement can contain **elif** clauses without a final **else** clause

Example: hiring.py

```
def decision(gpa, interview, test):
    points = 0                    # Variable to store the total points

    if gpa >= 3.3:
        points += 1

    if interview >= 9:
        points += 2
    elif interview >= 7:
        points += 1               # note: no else clause

    if test > 85:
        points = points + 1

    if points <= 2:
        return 'Not hired'
    elif points == 3:
        return 'Junior Salesperson'
    else:
        return 'Manager-in-Training'
```

Ranges and relational operators

The relational operators can be used to express ranges of values

Examples:

- An age in the range 1 through 25, inclusive:
  1 <= age <= 25
- A length in the range 15 (inclusive) up to, but not including, 27:
  15 <= length < 27
- A year in the range 1900 through 1972, exclusive of both:
  1900 < year < 1972

63

More on strings

Python strings can begin and end with single quotes or double quotes

- **'Stony Brook'** and **"Stony Brook"** are both valid ways of defining the same string

Recall that the plus symbol joins two strings into a single longer string (concatenation)

The asterisk repeats a string a specified number of times

- Example: **'Hello' * 3** will evaluate to **'HelloHelloHello'**

64

String functions

Strings are very fundamental to programming

- Most languages (including Python) support many functions and other operations for strings.

The Python function named **len** (short for "length") counts the number of characters in a string

- **len** counts every character in a string, including digits, spaces, and punctuation marks
- Example:

```
school = 'Stony Brook University'
n = len(school)        # n will equal 22
```


String methods

Many other functions on strings are called using a different syntax

Instead of writing **func(s)** they are written **s.func()**

- The name of the string is written first, followed by a period, and then the function name

Functions called using this syntax are referred to as **Methods**

String methods

As an example of a string method, consider how to figure out how many words are in a sentence

If there is exactly one space between each word, just count the number of space characters and add one

The method named count does exactly that:

```
sentence = 'It was a dark and stormy night.'
sentence.count(' ') + 1 # equals 7
```

Note the argument passed to count is a string containing exactly one character: a single space character.

String methods

Two other useful methods are **startswith** and **endswith**

- These are both Boolean functions and return **True** or **False** depending on whether a string begins or ends with a specified value

Examples:

```
sentence = 'It was a dark and stormy night.'
sentence.startswith('It')        # True
sentence.startswith('it')        # False
sentence.startswith("It's")      # False
sentence.endswith('?')           # False
sentence.endswith('.')           # True
```

String methods

Another example:

```
filename = input('Enter a filename: ')
if filename.endswith('.py'):
    print('The file contains a Python program.')
else:
    print('The file does not contain a Python program.')
```


Questions?