

# Introduction to Computers

LECTURE 2B – COMPUTER PROGRAMMING  
FUNDAMENTALS II

---

SLIDES COURTESY: PROF. ANTONINO MIONE (SUNY  
KOREA)

A solid orange horizontal bar spanning the width of the slide, located at the bottom.

# Announcements

---

This lecture: Computer Programming Fundamentals

Reading: Read Chapter 2 of Conery

**Acknowledgement:** Some of the lecture slides are based on CSE 101 lecture notes by Prof. Kevin McDonald at SBU and the textbook by John Conery.

# Functions

---

- Recall that Python has a math module
  - The library has numbers ( $e$ ,  $\pi$ , etc.)
  - It also has a variety of useful mathematical functions
- 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
- **print**, **int**, **float**, and **str** are also functions
- The next example introduces a new function, **format**, that lets the programmer format numerical output in a desired way

# 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:  
$$\text{BMI} = (\text{weight} * 703) / \text{height}^2$$
- A BMI in the range of 18.5-24.9 is considered “healthy”
- The next program being examined calculates and prints a person's BMI based on entered values
- The result is printed to 15 digits of accuracy, which is more digits than necessary

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

# 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) # cf. 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
```

```
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!



# 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

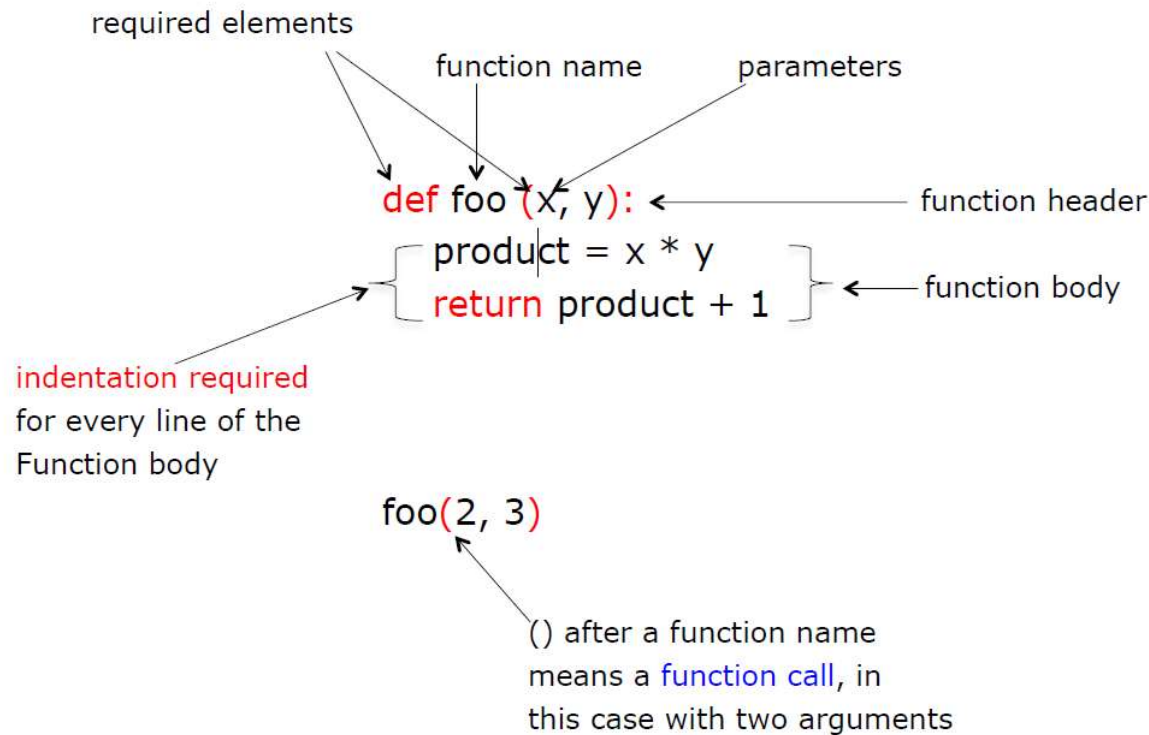
1. Define a function, once

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

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

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

# Mechanics of defining/calling a function



# Parameters and arguments

---

- Function can have **zero or more** parameters
  - Function may be defined with *formal parameters*
  - Then called with *actual arguments*
  - How many? As many as the function needs!

- 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))
```

# Program: flow of execution

---

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

Note: three functions and a call  
to message on the left is a *program*!

Output:

```
1  
a  
middle  
b  
2
```

# Void functions vs. fruitful 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
- **square** is an example of a *fruitful function*
  - It **returns** a value when it is called

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

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

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

```
print(square(3))
```

See what gets printed by the print statement in each case!

- Don't expect a void function to return any useful value.
  - The call `announce('hello')` returns **None** to indicate that.
- So the statement `print(announce('hello!'))` will print **None**.

# 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 calling this function will do a desired computation
- Functions thereby allow complex problems to be solved by subdividing it into smaller, more manageable sub-problems
  - This process is called **problem decomposition** (also **functional decomposition**)
- Often programmers use functions to engage in **top-down software design**
  - They design the software as a series of steps
  - Each step corresponds to one or more functions

# 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



# 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()
```

# Example: Distance calculator

---

- Example: A distance traveled is provided in miles, yards, and feet (i.e. 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

# 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:

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

- 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
- 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}'`
- Following is a function to compute pmt

# Example: mortgage.py

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

Disregard material on 'format()'. This will be covered in detail in a lab later in the semester.

# 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

# Example: tuition.py

---

```
def main():
    credits = int(input('Enter number of credits: '))
    tuition = getTuitionFees(credits)
    print('A student taking ' + str(credits) + ' will pay $' + str(tuition) + ' in tuition.')

def getTuitionFees(numCredits):
    cost = 0
    if numCredits < 12:
        cost = numCredits*600
    else:
        cost = 5000
    return cost

main()
```

# Function to distinguish between 3 tax brackets

---

- This function distinguishes between three tax brackets:

```
def marginal_tax_rate(income):
```

```
    if income < 10000:
```

```
        return 0.0
```

```
    elif income < 20000:
```

```
        return 5.0
```

```
    else:
```

```
        return 7.0
```



# 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

# Boolean expressions

---

The notation  $\geq$  means “greater than or equal to” and is one of six **relational operators** supported by Python:

Mathematical Operator	Python Equivalent	Meaning
$=$	<code>==</code>	is equal to
$\neq$	<code>!=</code>	is not equal to
$>$	<code>&gt;</code>	is greater than
$\geq$	<code>&gt;=</code>	is greater than or equal to
$<$	<code>&lt;</code>	is less than
$\leq$	<code>&lt;=</code>	is less than or equal to

# 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 or not an employee is entitled to overtime pay
  2. If so, how much?
- #1 is pretty simple: use an if-statement
- For #2, a different calculation is required depending on whether employee will earn overtime pay or not
- Regular pay formula:  $\text{hourly wage} \times \text{hours worked}$
- The overtime formula has two parts:
  - The pay for first 40 hours
  - The pay for additional overtime hours

# 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()
```

Disregard material on 'format()'. This will be covered in detail in a lab later in the semester.

# 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/features will help with this task:
  - The `+=` operator can be used to increment a variable by some amount
  - `-=`, `*=` and `/=` also exist and perform analogous operations
  - A variable can be used to maintain a tally or running total
  - An if-statement can contain **elif** clauses without a final **else** clause



# Example: hiring.py

---

```
def decision(gpa, interview, test):
    points = 0          # Point total accumulator

    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:

- `0 <= age <= 25`**

- A length in the range 15 (inclusive) through 27:

- `15 <= length < 27`**

- A year in the range 1900 through 1972, exclusive of both:

- `1900 < year < 1972`**

```
age = int(input("Enter the age:"))
```

```
if 0 <= age <= 25:
```

```
    print("age is in the range 0 and 25")
```

```
else:
```

```
    print ("age is outside range")
```



# 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'**

# String functions

---

- Strings are very fundamental to programming
  - most languages support many functions and other operations for strings
  - Python is no exception.
- 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
- 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?

---