

Just enough Python  
to get started with





# Python 3 and IDLE

- We will use version 3.  $x$  of Python (where  $x$  is the most recent version)
  - Differences between Python 2 and Python 3 are mostly minor, but can be confusing
- Python comes with an *IDE* ( *Integrated Development Environment* ) called *IDLE*
  - IDLE is a *REPL* ( *Read-Evaluate-Print-Loop* ) that lets you enter Python statements one at a time, and see what they do
  - IDLE also lets you create, edit, run, test, and debug programs
- **Note:** Anything in *red* is a jargon term which you are expected to learn

**Try it now!**



# Program components

- A program typically needs to:
  - **Read** information in from somewhere (the keyboard, or a file)
  - **Perform computations** on numbers, strings (text) and booleans (logical true/false values)
  - **Make decisions** , based on the current state of the program
  - **Repeat** the same operation over and over again
  - **Delegate** : Perform complex operations described separately and given appropriate names
  - **Write** out results to somewhere (the screen, or a file)



# Values

- There are many different kinds of values, including:
  - *integers* (“whole” numbers, such as **23** and **-5** )
  - *floating-point numbers* (numbers containing a decimal point, such as **3.1416**)
  - *strings* (sequences of characters enclosed in single or double quotes, such as **"hello"**)
  - *booleans* (one of the two values **True** and **False**)
- An explicit value, written out by itself, is called a *literal* or *literal value*
- Later we will meet more kinds of values, such as *lists* and *dictionaries* , which are composed of these simpler kinds of values



# Variables

- A *variable* is a name that “holds,” or is associated with, a *value*
- Values can be assigned to variables by means of an *assignment statement*
  - **Example:** `age = 23`
- The value of a variable can be changed (varied):
  - **Example:** `age = age + 1`
- In Python, the *type* (kind) of a variable can also be changed (this is not true in many languages):
  - **Example:**  
`age = "Old enough to know better!"`



# Reading input from the user

- A **function** is a named piece of code that can return a value
- The **input** function is used to read input from the user
  - There are two forms, with and without an **argument**:
    - **input()** just returns a string entered by the user
    - **input(prompt)** displays the “prompt” string, then returns the string entered by the user
    - **Example: name = input(“What is your name?”)**
    - Usually (as in this example) you will want to save the entered value in a variable
  - The value returned by input is always a string
  - If you want to read a number from the user, use the additional functions **int** or **float**
    - **Example: age = int(input(“What is your age?”))**

**Try it now!**





# Doing arithmetic

- Arithmetic is slightly complicated because there are two kinds of numbers, *integers* (“whole numbers”) and *floating-point numbers* or *floats* (numbers containing a decimal point)
- Operations are  $+$  (add),  $-$  (subtract),  $*$  (multiplication), two kinds of division,  $/$  and  $//$ , and  $\%$  (modulus, or remainder of a division)
  - When you use  $+$ ,  $-$ ,  $*$ ,  $//$ , or  $\%$  on **just integers**, you get an integer result
    - $//$  is called *integer division*
    - If the numbers don’t divide evenly, you get the smaller number as a result
  - All other combinations result of numbers and operations result in a float
- Parentheses  $()$ , but *not* brackets  $[]$  or braces  $\{\}$ , can be used to group operations

**Try it now!**



# Using strings

- A string is a sequence of characters enclosed in either single quotes `'...'` or double quotes `"..."`
- A string enclosed in single quotes may contain double quotes, and vice versa
- Some single characters cannot easily be entered directly into strings, and must be “escaped” (backslashed)
  - `\n` represents a newline character
  - `\t` represents a tab character
  - `\'` represents a single quote (inside a singly-quoted string)
  - `\"` represents a double quote (inside a doubly-quoted string)
- Strings can be concatenated (joined) with the `+` operator
- **Example:** `"Do you love me\nOr do you not?" + "You told me once\nBut I forgot."`
- So-called “triple quotes”, `"""..."""` or `'''...'''`, can be used to write strings that extend over multiple lines

**Try it now!**



# Using booleans

- The two *boolean* values are **True** and **False**
  - **Capitalisation** is important!
- The three boolean operators are **not**, **and**, and **or**
- The following comparison operators on numbers will give a boolean result:
  - <** (less than)
  - <=** (less than or equal)
  - ==** (equal)
  - !=** (not equal)
  - >=** (greater than or equal)
  - >** (greater than)
- These comparisons also work on strings (all capital letters **<** all lowercase letters)
- Booleans, like numbers and strings, can be assigned to variables
- **Example: `in_range = grade >= 0 and grade <= 100`**

**Try it now!**



# The `print` function

- In Python 3, `print` is a function, but is used like a statement
  - More about functions later
- **Syntax:** `print(arguments)`
  - The `arguments` are values, variables, or expressions, separated by commas
  - The `arguments` are “printed” (displayed on the screen) on a single line, separated by spaces
- **Example:** `print("You have", points, "points.")`
- **Note:** `print` statements are seldom used in the REPL, because they are built into the Read-Eval- **Print** -Loop, so results are printed automatically



# Control statements

- **Control statements** are used to decide whether and how often some other, “controlled” statements are executed
- **if** statements decide whether or not to execute a group of statements
- **if-else** statements decide which of two groups of statements to execute
- **while** statements execute a group of statements as long as some condition is true
- **for** statements execute a group of statements with a variable taking on a sequence of values
- For every kind of control statement:
  - The control statement ends in a colon, :
  - The controlled statements are indented four spaces
    - In IDLE, pressing the Tab key is the same as typing four spaces





# Layout

- Every statement goes on a line by itself
- Put spaces around operators, including the assignment operator, =
  - **average = sum / 5**
- Put spaces after commas (but not before commas)
  - **print(2, "plus", 2, "is", 2 + 2)**
- When using a function, do not put spaces on either side of the parentheses
  - **age = input("What is your age? ")**
- Do not put spaces inside parentheses
  - **age = input( "What is your age? " ) # Don't do this!**



# if statements

- The **syntax** of a language is the set of rules that things in the language must follow
- The syntax of an **if** statement is basically:  
**if** condition:  
    one or more statements
- **Example:**  
**if age < 18:**  
    **print("Too young to drink alcohol.")**
- **Explanation:**  
If the variable **age** holds a number less than 21, the message is printed; but if it holds a number equal to or greater than 21, these statements do nothing



# if-else statements

- The syntax of an **if-else** statement is basically:

**if** condition:

one or more statements

**else:**

one or more statements

- **Example:**

**if** age < 18:

**print("Go home, kid!")**

**print("You're too young to drink.")**

**else:**

**print("Yes, sir, how may I serve you?")**

- **Explanation:**

If the variable **age** is less than 18, the first two messages are printed; otherwise, the third message is printed

**Try it now!**



# if with elif

- The **if** statement can have any number of **elif** tests
- Only one group of statements is executed—those controlled by the first test that passes
- **Example:**
- **if** grade == "A":  
    **print "Congratulations!"**
- elif** grade == "B":  
    **print "That's pretty good."**
- elif** grade == "C":  
    **print "Well, it's passing, anyway."**
- else:**  
    **print "You really blew it this time!"**



# Control statements can be nested

- Control statements may be nested; that is, control statements may be controlled by other control statements

- **Example:**

```
if sex == "male":  
    if is_smoker:  
        life_expectancy = 72.3  
    else:  
        life_expectancy = 84.3  
else: # female  
    if is_smoker:  
        life_expectancy = 75.6  
    else:  
        life_expectancy = 86.6
```

- Compound variable names, such as `is_smoker`, are usually written with an underscore between the words
- The `#` symbol introduces a **comment**—ignored by the computer, but intended to help the human reader



# while loops

- A “while loop” has this syntax:

**while** condition:

    one or more statements

- Here's what it does:

1. Check the condition

2. If it is **True**:

    execute the statements

    start the loop again (at 1)

3. If it is **False**:

    exit the loop

- **Example:**

```
countdown = 10
```

```
while countdown >= 0:
```

```
    print(countdown)
```

```
    countdown = countdown - 1
```

```
print("Blast off!")
```

**Try it now!**





# for loops

- A “for loop” has this syntax:

```
for variable in sequence :  
    one or more statements
```

- One way to get a sequence is to list the members of the sequence in brackets, [], separated by commas

```
for word in [“one”, “two”, “three”]:  
    print(word)
```

- Another way is to use the range function, **range**(start, end ) which will return a sequence of integers from start up to, but not including, end

```
for number in range(1, 11):  
    print(number)
```

- This prints the numbers 1 through 10, each on a separate line

**Try it now!**



# Functions

- A *function* is a named chunk of code that can be executed (“*called*”) by giving its name and whatever arguments it requires
- The *arguments* are matched up to the *parameters*
  - There must be the same number of arguments as there are parameters, and they are matched up by position
- A function may (or may not) *return* a value
- **Syntax:**
  - def** function\_name( parameters ):
    - Optional string describing the function, usually triple-quoted
    - One or more statements
- Functions are sometimes called *methods*
  - For now, treat the two words as synonyms; we will explain the difference later



# Function example

- **Example function definition:**

```
def lcd(a, b):  
    """Compute largest common divisor of a and b"""  
    while b != 0:  
        r = a % b  
        a = b  
        b = r  
    return a
```

- This function has two parameters, so it should be called with two arguments
- Since the parameters are treated as numbers, the arguments to it should be numbers (or variables containing numbers)
- **Example function call:**

```
print("The LCD of", 12, "and", 5, "is", lcd(12, 5))
```



# Programs

- A *program* is code that has been saved to a file
  - The file should have the **.py** extension
  - You can create a new file in IDLE, or load in an existing file
  - The file can be executed by hitting the **F5** key
- A program is executed as it is loaded in, top to bottom.  
It can be either:
  - Just a collection of statements, executed one after the other, or
  - A collection of functions that can be called individually from the REPL, or
  - A collection of functions, plus special code to start the program and call the various functions as needed. The special code is:

```
if __name__ == "__main__":
```

Statements to execute when the program is loaded

**Try it now!**



# Errors

- **Errors are inevitable.** You will make mistakes. If this embarrasses you, *get over it!*
- Most of computer science is learning how to minimize errors, find them when they occur, and recover from them
- Kinds of errors:
  - A **syntax error** is one recognized by the **compiler** (the thing that gets your program ready to execute), and prevents it from even starting  
Example: `print('This won't work')`
  - A **runtime error** is one that causes your program to “crash”  
Example: `y = 3 / (x - x)`
  - A **logic error** or **semantic error** is one that causes your program to produce incorrect results  
Example: `hypotenuse = math.sqrt(a * a + b + b)`
  - A **user error** is when the user provides invalid input to the program, causing the program to crash or to produce incorrect results
    - We will discuss how to handle user errors in a later lecture



# Comments

- A *comment* is a note to any human looking at the program; comments are ignored by the computer.
- A comment begins with `#` and extends to the end of the line
- Good uses of comments:
  - At the beginning of a program, to tell what the program does
  - When using someone else's code, to say where you got it from
  - To explain any code that's hard to understand
- Bad uses of comments:
  - To explain something that's obvious anyway
  - To explain code that's hard to understand, but could be made simpler
  - To add irrelevant comments, like `# Go Eagles!`
  - When you should instead use a *doc string* (the string describing what a function does )





# A Complete Sample Program

```
def main():
    """Converts between Fahrenheit and Celsius"""
    f_to_c = mode()
    print("OK. Enter Q or q to quit.")

    t = input("\nEnter a temperature: ")
    while t != 'Q' and t != 'q':
        t = float(t)
        if f_to_c:
            print(t, "F is", 5 * (t - 32) / 9, "C")
        else:
            print(t, "C is", 1.8 * t + 32, "F")
        t = input("\nEnter a temperature: ")
    print("Done.")

def mode():
    print("Type F to convert TO Fahrenheit,")
    answer = input("C to convert TO Celsius: ")
    if answer == 'C':
        return True
    elif answer == 'F':
        return False
    else:
        return mode()

if __name__ == "__main__":
    main()
```

**Try it now!**



“ Programming is an art form that fights back.”  
-- Anonymous