Starting with Python

CORE-UA 109.01, Joanna Klukowska
adapted from slides for CSCI-UA.002 by D. Engle, C. Kapp and J. Versoza

printing or showing information on the screen

`print()` function

```
print ("hello core109 - congratulations on running your first program" )
print ( 3 * 17 - 102/3 + 7 )
```

- `print` does what you might expect it to do: it prints some content to the screen/console
- `print` takes either a string (first case above) or a numerical value (second case above) as its arguments
  - strigs are sequences of characters that start and end eihter with a single quote ' or with a double quote "
  - when the numerical value given to the `print` function is a more complicated expression, the `print` function evaluates it first before the value is printed (so the `print` function first figures out what the value of `3 * 17 - 102/3 + 7` is and then prints 24)

more on `print()` function

Can I print more than one *thing* using `print()`?

Sure ... just concatenate your *things* with a comma, like this:

more on `print()` function

Can I print more than one *thing* using `print()`?

Sure ... just concatenate your *things* with a comma, like this:

```
print ("my name is", "Joanna")
```

This will concatenate the two strings first and then print the resulting string.

more on `print()` function

Can I print more than one *thing* using `print()`?

Sure ... just concatenate your *things* with a comma, like this:

```
print ("my name is", "Joanna")
```

This will concatenate the two strings first and then print the resulting string.

```
print ("The value of  3 * 17 - 102/3 + 7  is ",  3 * 17 - 102/3 + 7)
```

This will concatenate the first string that contains a mathematical expression with the value of 24.0 that results from calculating expression after the comma.

more on `print()` function

Can I print more than one *thing* using `print()`?

Sure ... just concatenate your *things* with a comma, like this:

```
print ("my name is", "Joanna")
```

This will concatenate the two strings first and then print the resulting string.

```
print ("The value of  3 * 17 - 102/3 + 7  is ",  3 * 17 - 102/3 + 7)
```

This will concatenate the first string that contains a mathematical expression with the value of 24.0 that results from calculating expression after the comma.

Can I concatenate more than two *things*?

Absolutely!

... just don't overdo it, otherwise your code becomes harder to read.

`print()` examples

```
print("Line 1")
```

```
Line 1
```

This is just a regular print statement and its output.

`print()` examples: `end=''`

```
print('Line 2, part 1')
print('Line 2, part 2')
```

```
Line 2, part 1
Line 2, part 2
```

I wanted for these two strings to be printed on a single line!

What happened?

`print()` examples: `end=''`

```
print('Line 2, part 1')
print('Line 2, part 2')
```

```
Line 2, part 1
Line 2, part 2
```

I wanted for these two strings to be printed on a single line!

What happened?

- **`print`** function automatically adds a newline character at the end of the printed text
- we can change that by telling it to use a different end of line character(s)

```
print('Line 2, part 1', end='')
print('Line 2, part 2')
```

```
Line 2, part 1Line 2, part 2
```

```
print('Line 2, part 1', end='***')
print('Line 2, part 2')
```

```
Line 2, part 1***Line 2, part 2
```


`print()` examples: `sep=''`

```
print("My name is ", "Joanna.")
print("I am learning", "Python.")
```

```
My name is  Joanna
I am learning Python
```

How do these two statements differ?

`print()` examples: `sep=''`

```
print("My name is ", "Joanna.")
print("I am learning", "Python.")
```

```
My name is  Joanna
I am learning Python
```

How do these two statements differ?

- the first one has an extra space between "is" and "Joanna"
- **`print`** adds spaces between the elements of the comma-separated lists of its arguments automatically

We can change this behavior by specifying a different separator:

```
print("My name is ", "Joanna.", sep="")    #do not use any separator
print("My name is", "Joanna.", sep="***")  #use three stars as a separator
print("I am learning", "Python.", sep="  !  ")  #use spaces and a ! as a separator
```

```
My name is Joanna.
My name is***Joanna.
I am learning  !  Python.
```

`print()` examples: `sep=''`

```
print("My name is ", "Joanna.")
print("I am learning", "Python.")
```

```
My name is  Joanna
I am learning Python
```

How do these two statements differ?

- the first one has an extra space between "is" and "Joanna"
- `print` adds spaces between the elements of the comma-separated lists of its arguments automatically

We can change this behavior by specifying a different separator:

```
print("My name is ", "Joanna.", sep="")    #do not use any separator
print("My name is", "Joanna.", sep="***")  #use three stars as a separator
print("I am learning", "Python.", sep="  !  ")  #use spaces and a ! as a separator
```

```
My name is Joanna.
My name is***Joanna.
I am learning  !  Python.
```

You can can use `end=` and `sep=` together.

variables

variables

- Variables are like little "buckets" or "containers" that can store information.
- You can create a variable by using the following syntax:

  `variablename = somedata`

  Examples:

  ```
  speed = 5
  myname = 'Kate'
  ```

- The '=' symbol is called the **assignment operator** and will cause Python to store the data on the right side of the statement into the variable name printed on the left side.

variables - example

```
num1 = 57   #assign a value of 57 to variable named num1
num2 = 89   #assign a value of 89 to variable named num2

#calculate the sum of the values stored in num1 and num2
sum_of_nums = num1 + num2
```

- this program does not have any output (why?)
- first the values are saved in variables called `num1` and `num2` and then the sum of those values is saved in another variable called `sum_of_nums`

why variables?

```
num1 = 57   #assign a value of 57 to variable named num1
num2 = 89   #assign a value of 89 to variable named num2

#calculate the sum of the values stored in num1 and num2
sum_of_nums = num1 + num2
```

Why do we need those variables instead of just using the numbers themselves to calculate the sum?

why variables?

```
num1 = 57   #assign a value of 57 to variable named num1
num2 = 89   #assign a value of 89 to variable named num2

#calculate the sum of the values stored in num1 and num2
sum_of_nums = num1 + num2
```

Why do we need those variables instead of just using the numbers themselves to calculate the sum?

- we might want to perform other operations on the same numbers - dealing with variables guarantees (almost) that we can use the same exact values for each operation, for example

  ```
  diff_of_nums = num1 - num2
  product_of_nums = num1 * num2
  ```

- we might want to change the value of the number that we are working with, for example,
  - after we wrote the entire program using 57 in thirteen different places, we realized that we really wanted to use 58
  - if the value is stored in a variable, we only need to change one line of code; if not, we have to search for all different locations of 57 and replace them with 58

Variable naming rules

- You can't use one of Python's **reserved words**

  `False`, `None`, `True`, `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `try`, `while`, `with`, `yield`

- Variables can't contain spaces (though you can use the underscore character ("_") in place of a space)
- The first character of a variable name must be a letter or the underscore character. Any character after the first can be any valid alphanumeric character (or the underscore character)
- Python is case sensitive, so `num` and `Num` are two different variable names

getting information from the user

`input()` function

```
name = input("Hello! What is your name? ");
print ("Hi " + name + "! It is lovely to meet you!\n");

num_of_programs_written = input ("How many programs have you written " +
        "in your life? ")
print ("Wow " + num_of_programs_written +
        "!!! That's a good number.")
```


`input()` function

```
name = input("Hello! What is your name? ");
print ("Hi " + name + "! It is lovely to meet you!\n");

num_of_programs_written = input ("How many programs have you written " +
        "in your life? ")
print ("Wow " + num_of_programs_written +
        "!!! That's a good number.")
```

- the `input()` function takes **a prompt** as its argument - it is simply used to tell the user what it is that the program wants
- the `input()` function **returns** (or gives back to the program) **the sequence of characters** (also known as **a string**) that the user typed in response to the prompt
- the value returned by the `input()` function can be saved in **a variable**: `name` and `num_of_programs_written` in the example above
- this is another reason why we use variables - they store the values returned by functions

`input()` function with numbers

Challenge:

```
fav_num = input("What is your favority number? ")

print("Cool, I like ", fav_num, " as well.")
print("I also like twice that value:", 2 * fav_num)
print("and I really like one more than that number:", 1 + fav_num)
```

What happens when the above program runs and the user enters 7?

`input()` function with numbers

Challenge:

```
fav_num = input("What is your favority number? ")

print("Cool, I like ", fav_num, " as well.")
print("I also like twice that value:", 2 * fav_num)
print("and I really like one more than that number:", 1 + fav_num)
```

What happens when the above program runs and the user enters 7?

```
What is your favority number? 7
Cool, I like  7  as well.
I also like twice that value: 77
Traceback (most recent call last):
  File "/home/joannakl/week1/favorite_number.py", line 5, in <module>
    print("and I really like one more than that number:", 1 + fav_num)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

What went wrong?

How do we fix it?

`input()` function with numbers

```
fav_num = input("What is your favority number?")

print("Cool, I like ", fav_num, " as well.")
print("I also like twice that value:", 2 * fav_num)
print("and I really like one more than that number:", 1 + fav_num)
```

What is the value returned by the `input()` function?

- you may think that it is `7` - the number that the user entered
- ... but, remember that this function returns a sequence of characters that the user entered, not a number
- the actual return value is `"7"` - a string containing the character 7
- **we cannot do arithmetic on strings** - at least not the kind of arithmetic that we had in mind

`input()` function with numbers

```
fav_num = input("What is your favority number?")

print("Cool, I like ", fav_num, " as well.")
print("I also like twice that value:", 2 * fav_num)
print("and I really like one more than that number:", 1 + fav_num)
```

How do we tell the program that we want to use the value entered by the user as a number, not as a string?

- we use `int()` function to convert a value to an integer (**Integers** are positive and negative whole numbers)

```
fav_num = int ( input("What is your favority number?") )

print("Cool, I like ", fav_num, " as well.")
print("I also like twice that value:", 2 * fav_num)
print("and I really like one more than that number:", 1 + fav_num)
```

```
What is your favority number? 7
Cool, I like  7  as well.
I also like twice that value: 14
and I really like one more than that number: 8
```


conversion functions

`int()`, `str()`, `float()`

- these functions all return a value or cause an error
- `int( x )` - converts a string or a number to an integer
- `str ( x )` - converts the argument to a string
- `float ( x )` - converts a string or a number to a floating point number

int(), str(), float()

- these functions all return a value or cause an error
- `int( x )` - converts a string or a number to an integer
- `str ( x )` - converts the argument to a string
- `float ( x )` - converts a string or a number to a floating point number

```
n1 = int("45")          # returns a number 45
n2 = int(-78.990)       # returns a number -78
n3 = int("thirteen")    # causes an error !!!

f1 = float("-56.90876") # returns a floating point number -56.90876
f2 = float( 39 )        # returns a floating point number 39.0
f3 = float("4.0 GPA")   # causes an error
```


operators:
+, -, *, /, **

arithmetic operators: +, -, *, /, **

with numbers (whole numbers and decimal numbers) the above operators are used to perform standard mathematical operations:

- + addition, for example `5.6 + 19`
- - subtraction, for example `5.6 - 19`
- * multiplication, for example `5.6 * 19`
- / division, for example `5.6 / 19`
- ** exponentiation, for example `5.6 ** 19`

```
n1 = 5.6
n2 = 19

print (n1, " + ", n2, " = ", n1 + n2 )
print (n1, " - ", n2, " = ", n1 - n2 )
print (n1, " * ", n2, " = ", n1 * n2 )
print (n1, " / ", n2, " = ", n1 / n2 )
print (n1, " ** ", n2, " = ", n1 ** n2 )
```

```
5.6  +  19  =  24.6
5.6  -  19  =  -13.4
5.6  *  19  =  106.39999999999999
5.6  /  19  =  0.29473684210526313
5.6  **  19  =  164275392349701.66
```


string operators: +, *

with string the above operators are used to perform concatenation and replication:

- + concatenation
  - combine two strings into a single one
  - both of the sides of the + operators need to be strings
- * replication
  - one operand is a string, the other is a positive integer
  - the string is repeated as many times as the number suggests

```
str1 = "crazy "
str2 = "hats"
str3 = str1 + str2
str4 = 4 * str1
str5 = str2 * 3

print (str3 )
print (str4 )
print (str5 )
```

```
crazy hats
crazy crazy crazy crazy
hatshatshats
```

summary

string - summary

- Data that is textual in nature (i.e. "Hello, World!") is called a "String"
- Strings can contain 0 or more printed characters
- **String Literals** are strings that you define inside your program. They are "hard coded" values and must be **delimited** using a special character so that Python knows that the text you've typed in should be treated as printed text (and not a function call, variable name, etc.)
  Ex: `print ('hello, world!')`
- Python supports three different delimiters
  - The single quote/"tick" (')
  - The double quote (")
  - The triple quote (""")

functions - summary

- A "function" is **a pre-written piece of computer code** that will perform a specific action or set of actions
- Python comes with a number of **built-in functions**, and you can also write your own (more on that later in the semester)
- Functions always begin with a keyword (the **name of the function**) followed by a pair of parenthesis. Ex: `print()`, `input()`
- You can **pass one or more arguments** into a function by placing data inside the parenthesis Ex: `print('Hello, World!')`
  - Different functions "expect" different arguments. The print function, for example, expects printed text as an argument
- Functions may or may not **return a value**.
  - Some functions simply perform some action. Ex. `print()`
  - Other functions perform a computation and return its result to the user. Ex. `input()`
- When you ask Python to run a function we say that you have **called the function**.

variables - summary

- Variables are like little "buckets" that can store information in your computer's memory
- You will be using variables constantly when writing your programs in order to keep track of various pieces of information
- You can create a variable by using the following syntax:

  ```
  variablename = somedata
  ```

  Examples:

  ```
  speed = 5
  ```

  ```
  myname = 'Craig'
  ```

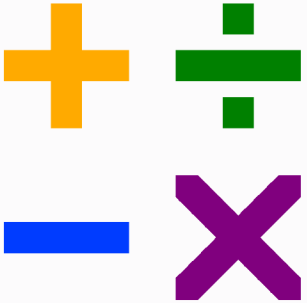
- The = symbol is called the **assignment operator** and will cause Python to store the data on the right side of the statement into the variable name printed on the left side

programming challenges

Mini-calculator

- Ask the user for two numbers. You can assume they will be floating point numbers.
- Compute the following and print it out to the user:
  - The sum of the numbers
  - The product of the numbers
  - The difference between the numbers
  - The first number divided by the second number
  - The first number raised to the power of the second number
- Display the mathematical expression and the result, for example, if the user enters 35 and 7, the first line of output for your program should be

```
35 + 7 = 42
```



Subway ride calculator

- Write a program that asks the user for the value of their current Metro card
- Compute how many rides they have left on their card. Only provide whole number results (i.e. you cannot have 3.5 rides left on a card)

Hints:

- A single ride costs $2.75 these days.
- Conversion functions will be useful here. Remember that
  - `float("25.75")` function will return the numberic value of `25.75`,
  - `int(3.5)` will return the integer part of its argument, i.e., `3`.

Coins

- Write a program that asks the user for a number of pennies, nickels, dimes and quarters
- Calculate the total amount of money that the user has and print it out

For this program, you can assume that the user will always enter a valid answer (i.e. non-negative integer).