Graphics and animation

Pixar and Video Games

Animated movies and **video games** today rely heavily on 2D and 3D graphics and animation technology.

- Pixar
- Disney
- Minecraft
- ...

Unit 3 Graphics and Animation

1/APPLICATIONS

In this unit, we'll learn about the computing science field of graphics and animation.

2/ALGORITHMS

We'll learn about turtle graphics, functions, the while loop, and recursion (2nd week).

3/PROGRAMMING LANGUAGE

In Python, we'll be learning the syntax and keywords to implement our algorithms.

4/DOCUMENTATION AND TESTING

We will show how to tell if our program is any good or not!

Question 1

What will the code below output?

```
word = "spaghetti"

print(word.upper())

print(word)

word = word.upper()

print(word)
```

Question 2

What will happen when executing this code? Is there a problem?

```
file = open("jokes.csv")

for line in file:

    data = file.readline()
```

A Drawing Program

Turtle Intro

```
1   # Use the drawing Turtle to draw to the screen
2
3   # Import turtle package
4   import turtle
5
6   # Create our turtle named Anna!
7   # (You can name your turtle anything)
8   anna = turtle.Turtle()
9
10  # Move forward 100 pixels
11  anna.forward(100)
12
13  # Stamp
14  anna.stamp()
15
16  # Turn left 90', move forward and stamp
17  anna.left(90)
18  anna.forward(100)
19  anna.stamp()
20
21  # Hold the window open until we click
22  turtle.Screen().exitonclick()
```

anna is an "**object**" of type Turtle

Drawing a cookie

Some Turtle Functions

1. anna.penup()
   anna.pendown()
2. anna.goto()
3. anna.circle()
4. anna.stamp()
5. anna.forward(100)
   anna.backward(100)
6. anna.left(90)
   anna.right(190)

```
# Draw a cookie

# Make our turtle
import turtle
anna = turtle.Turtle()

# Draw the cookie outside
anna.penup()
anna.goto(-5, -30)
anna.pendown()
anna.circle(30)
anna.penup()

# Chocolate chip in the middle
anna.goto(0,0)
anna.stamp()

# Chocolate chip top left
anna.goto(-10, 10)
anna.stamp()

# Chocolate chip top right
anna.goto(10, 10)
anna.stamp()

# Chocolate chip bottom left
anna.goto(-10, -10)
anna.stamp()

# Chocolate chip bottom right
anna.goto(10, -10)
anna.stamp()

# Hold the window open until we click
turtle.Screen().exitonclick()
```

But what if I want multiple cookies?

Would I just copy paste the cookie code and change the values?

⇒ That would be a lot of repeated code. What can we do?

Functions

A sequence of steps with a name

http://interactivepython.org/runestone/static/thinkcspy/Functions/functions.html
http://interactivepython.org/runestone/static/thinkcspy/Functions/ATurtleBarChart.html
https://hourofpython.trinket.io/a-visual-introduction-to-python#/functions/functions-are-recipes

You've used functions before

```
21
22  # Chocolate chip top left, stamp
23  anna.goto(-10, 10)
24  anna.stamp()
25
```

goto is a function *that takes* 2 arguments/parameters

stamp is a function *that takes* 0 arguments/parameters

Build your own function

1. Use the keyword **def**
2. Choose a **name** for the function (convention is to use all lowercase, with underscores)
3. **Indent** the code of the function

This is called "**defining** a function".

```
# Draw a cookie using functions

# Make our turtle
import turtle
anna = turtle.Turtle()

# Define a cookie drawing function
def draw_cookie():
    # Draw the cookie outside
    anna.penup()
    anna.goto(-5, -30)
    anna.pendown()
    anna.circle(30)
    anna.penup()

    # Chocolate chip in the middle
    anna.goto(0,0)
    anna.stamp()

    # Chocolate chip top left
    anna.goto(-10, 10)
    anna.stamp()

    # Chocolate chip top right
    anna.goto(10, 10)
    anna.stamp()

    # Chocolate chip bottom left
    anna.goto(-10, -10)
    anna.stamp()

    # Chocolate chip bottom right
    anna.goto(10, -10)
    anna.stamp()
```

How to call a function

1. Just write the name of the function (without the def) along with parentheses
2. Include arguments if necessary

Note: you need to define the function BEFORE calling it

```
3   # Make our turtle
4   import turtle
5   anna = turtle.Turtle()
6
7   # Define a cookie drawing function
8   def draw_cookie():
9       # Draw the cookie outside
10      anna.penup()
11      anna.goto(-5, -30)
12      anna.pendown()
13      anna.circle(30)
14      anna.penup()
15
16      # Chocolate chip in the middle
17      anna.goto(0,0)
18      anna.stamp()
19
20      # Chocolate chip top left
21      anna.goto(-10, 10)
22      anna.stamp()
23
24      # Chocolate chip top right
25      anna.goto(10, 10)
26      anna.stamp()
27
28      # Chocolate chip bottom left
29      anna.goto(-10, -10)
30      anna.stamp()
31
32      # Chocolate chip bottom right
33      anna.goto(10, -10)
34      anna.stamp()
35
36  # Call (use) the draw cookie function
37  draw_cookie()
```

This is a function!

(But it's not so useful yet. Why?)

But this draws my cookies in the same spot!

⇒ Introducing **functions** with **arguments/parameters**.

Adding parameters

1. Add the parameters to the function **declaration**
2. Use the parameters in the **body** of the function

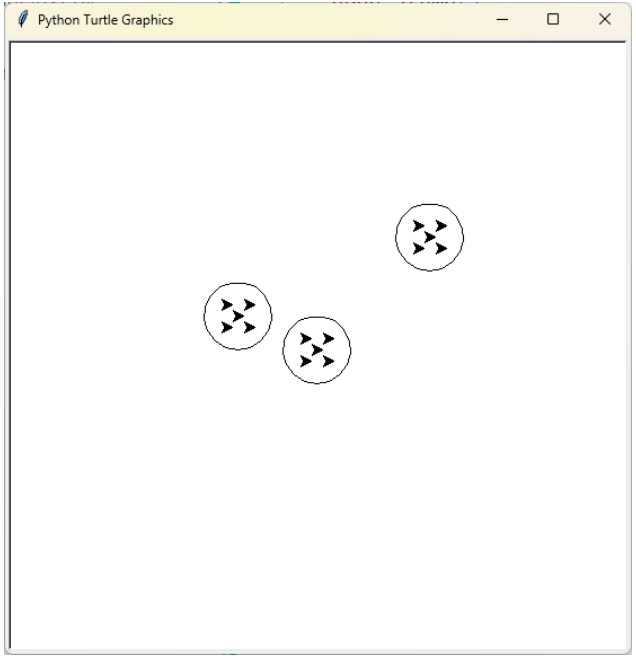
Now, when you call the function, you can now **vary** your parameters easily!

```
3   # Make our turtle
4   import turtle
5   anna = turtle.Turtle()
6
7   # Define a cookie drawing function
8   # Inputs: x - horizontal centre of cookie
9   #         y - vertical centre of cookie
10  def draw_cookie(x, y):
11      # Draw the cookie outside
12      anna.penup()
13      anna.goto(x - 5, y - 30)
14      anna.pendown()
15      anna.circle(30);
16      anna.penup()
17
18      # Chocolate chip in the middle
19      anna.goto(x, y)
20      anna.stamp()
21
22      # Chocolate chip top left
23      anna.goto(x - 10, y + 10)
24      anna.stamp()
25
26      # Chocolate chip top right
27      anna.goto(x + 10, y + 10)
28      anna.stamp()
29
30      # Chocolate chip bottom left
31      anna.goto(x -10, y - 10)
32      anna.stamp()
33
34      # Chocolate chip bottom right
35      anna.goto(x + 10, y - 10)
36      anna.stamp()
37
38  # Call (use) the draw cookie function
39  draw_cookie(0, 0)
40  draw_cookie(100, 100)
41  draw_cookie(-70, 30)
```

Function declaration (Line 10)

Function body

```
3   # Make our turtle
4   import turtle
5   anna = turtle.Turtle()
6
7   # Define a cookie drawing function
8   # Inputs: x - horizontal centre of cookie
9   #         y - vertical centre of cookie
10  def draw_cookie(x, y):
11      # Draw the cookie outside
12      anna.penup()
13      anna.goto(x - 5, y - 30)
14      anna.pendown()
15      anna.circle(30);
16      anna.penup()
17
18      # Chocolate chip in the middle
19      anna.goto(x, y)
20      anna.stamp()
21
22      # Chocolate chip top left
23      anna.goto(x - 10, y + 10)
24      anna.stamp()
25
26      # Chocolate chip top right
27      anna.goto(x + 10, y + 10)
28      anna.stamp()
29
30      # Chocolate chip bottom left
31      anna.goto(x -10, y - 10)
32      anna.stamp()
33
34      # Chocolate chip bottom right
35      anna.goto(x + 10, y - 10)
36      anna.stamp()
37
38  # Call (use) the draw cookie function
39  draw_cookie(0, 0)
40  draw_cookie(100, 100)
41  draw_cookie(-70, 30)
```



Try using various parameter values.

Variable Scope

Variable scope in functions

```
1   # Define a function
2   def print_power(a, b):
3       # Create a local variable, named `ans`
4       ans = a ** b
5       print(ans)
6
7   # Call
8   print_power(2, 3)
9
10  # Try to access print_power's local variable
11  # ** BAD **
12  print(ans)
```

ans is a local, temporary variable.

It is destroyed once the function call is complete

```
8
Traceback (most recent call last):
  File "c:\all-my-code\CMPT120-Code\ExWeek8\scope.py", line 12, in <module>
    print(ans)
          ^^^
NameError: name 'ans' is not defined. Did you mean: 'abs'?
```

Another example

```
def draw_square(side):
    sideLonger = side + 100
    for i in range(4):
        anna.forward(sideLonger)
        anna.left(90)
```

Local variables?

Let's review some concepts

How do we create a Turtle object, given the turtle module?

What is the benefit of defining a function?

What are function parameters or arguments good for?

What is the keyword necessary to create a function?

List vs Dictionary

Dictionary

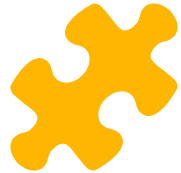
- In a list, we access elements by **a number (their index).**
- For example, how many people like Monday?

```
my_list = [0,0,0,0,0,0,0]
my_list[1] = 42
print(my_list[1])
```

- In a dictionary, we access elements by **a key, which can be anything!**
- For example, how many people like Monday?

```
my_dictionary = {
    "sun": 0,
    "mon": 0,
    "tue": 0
}
my_dictionary["mon"] = 42
print(my_dictionary["mon"])
```

Handy dictionaries



```
1   # Use a dictionary to get word translations
2   translations = {
3       "hello": "bonjour",
4       "bye":   "au revoir"
5   }
6
7   # Lookup a word:
8   word = translations["bye"]
9   print("Bye is: " + word)
10
11  # Print full dictionary
12  print("Full dictionary: ", translations)
```

Key

Value

Key

Value

```
Bye is: au revoir
Full dictionary:  {'hello':
'bonjour', 'bye': 'au revoir'}
```

We can **access elements** of the dictionary in a similar way to a list, but using the **key**. You cannot index a dictionary using 0,1,2 etc. in the same way you do lists.

Adding to dictionaries

```python
# Use a dictionary to get word translations
translations = {
    "hello": "bonjour",
    "bye":   "au revoir"
}

# Lookup a word:
word = translations["bye"]
print("Bye is: " + word)

# Print full dictionary
print("Full dictionary: ", translations)

# Add new word
translations["good"] = "bien"
print("Added good: ", translations)
```

```
Bye is: au revoir

Full dictionary:  {'hello':
'bonjour', 'bye': 'au revoir'}

Added good:  {'hello': 'bonjour',
'bye': 'au revoir', 'good': 'bien'}
```

Add to dictionaries like this

Iterating over dictionaries

```python
# Store and print the number of health workers per city.

# Dictionary of health workers per city
worker_locations = {
    "Freetown": 210,
    "Koidu": 50,
    "Kambia": 3,
    "Benguema": 45,
    "Makeni": 98,
}

# Print number of workers per city
for city in worker_locations:
    num_workers = worker_locations[city]
    print(f"In {city} there are {num_workers} workers.")
```

```
In Freetown there are 210 workers.
In Koidu there are 50 workers.
In Kambia there are 3 workers.
In Benguema there are 45 workers.
In Makeni there are 98 workers.
```

Same as worker_locations.keys()

What's wrong with this code?

```python
# Tally up favourite icecreams

tally_chocolate = 0
tally_vanilla = 0
tally_strawberry = 0

for i in range(5):
    choice = input("What is your favourite icecream? ").lower().strip("!. ")

    if choice == "chocolate":
        tally_chocolate += 1
    if choice == "vanilla":
        tally_vanilla += 1
    if choice == "strawberry":
        tally_strawberry += 1

print(f"Tallies:")
print(f"  Chocolate:  {tally_chocolate:3}")
print(f"  Vanilla:    {tally_vanilla:3}")
print(f"  Strawberry: {tally_strawberry:3}")
```

How does this code compare?

Hint: How much code is repeated?
Hint: What if the user types in "lemon"?
Hint: How easy is it to add new flavours?

Re-write this code as Dictionary

```
1   # Tally up favourite icecreams
2
3   tally_chocolate = 0
4   tally_vanilla = 0
5   tally_strawberry = 0
6
7   for i in range(5):
8       choice = input("What is your favourite icecream? ").lower().strip("!. ")
9
10      if choice == "chocolate":
11          tally_chocolate += 1
12      if choice == "vanilla":
13          tally_vanilla += 1
14      if choice == "strawberry":
15          tally_strawberry += 1
16
17  print(f"Tallies:")
18  print(f"  Chocolate:  {tally_chocolate:3}")
19  print(f"  Vanilla:    {tally_vanilla:3}")
20  print(f"  Strawberry: {tally_strawberry:3}")
```

Write a program to do the same task but use a dictionary instead.

Hint: Dictionary may look like:

```
tally = {
    "chocolate": 0,
    "vanilla": 0,
    "strawberry": 0
}
```

Adding to a specific dictionary value

```
# Tally up favourite ice creams

tally = {
    "chocolate": 0,
    "vanilla": 0,
    "strawberry": 0
}

for i in range(5):
    choice = input("What is your favourite ice cream? ").lower().strip("!. ")

    if choice in tally:
        tally[choice] += 1

print(f"Tallies:")
for key in tally:
    print(f"{key:>15}: {tally[key]:3}")
```

How does this code compare?
Hint: How much code is repeated?
Hint: What if the user types in "lemon"?
Hint: How easy is it to add new flavours?

We don't need to use a big conditional block!

We don't have to handle multiple variables!

Using dictionaries

```python
# Tally up favourite ice creams

tally = {
    "chocolate": 0,
    "vanilla": 0,
    "strawberry": 0
}

for i in range(5):
    choice = input("Ice cream? ").lower().strip("!. ")

    if choice in tally:
        tally[choice] += 1
    else:
        print("I don't know that flavour.")

print(f"Tallies:")
for key in tally:
    print(f"{key:>15}: {tally[key]:3}")
```

Same as tally.keys() when checking with in

Handling potential runtime error!

Same as tally.keys() when used in a for loop

Using dictionaries

```
1   # Tally up favourite ice creams
2
3   tally = {
4       "chocolate": 0,
5       "vanilla": 0,
6       "strawberry": 0
7   }
8
9   for i in range(5):
10      choice = input("Ice cream? ").lower().strip("!. ")
11
12      if choice in tally:
13          tally[choice] += 1
14      else:
15          tally[choice] = 0
16
17  print(f"Tallies:")
18  for key in tally:
19      print(f"{key:>15}: {tally[key]:3}")
```

Even better! Adding to the dictionary.

Count fav-days: Dictionary

```
1   # Find everyone's favourite day (using dictionary)
2
3   # Dictionary to map each day to a tally
4   fav_day_tallies = {
5       "sat": 0,
6       "sun": 0 }
7
8   for i in range(3):
9       # Ask user their favourite day
10      print("What is your favorite day?")
11      fav = input(": ").lower().strip()
12
13      # Add to day's tally
14      fav_day_tallies[fav] += 1
15
16  # Summary
17  for day in fav_day_tallies:
18      tally = fav_day_tallies[day]
19      print(f"{day:5} = {tally:>3}")
```

Directly access the day!

Count fav-days: Parallel Lists

Must find index (i) for the day, and then access 2nd array using [i]

```
1   # Find everyone's favourite day (using lists)
2
3   # Parallel arrays storing days and tallies
4   days = ["sat", "sun"]
5   fav_day_tallies = [0] * len(days)
6
7   for i in range(3):
8       # Ask user their favourite day
9       print("What is your favorite day?")
10      fav = input(": ").lower().strip()
11
12      # Add to day's tally
13      for i in range(len(days)):
14          if days[i] == fav:
15              fav_day_tallies[i] += 1
16
17  # Summary
18  for i in range(len(days)):
19      day = days[i]
20      tally = fav_day_tallies[i]
21      print(f"{day:5} = {tally:>3}")
```

Dictionary vs Parallel Lists

```python
# Find everyone's favourite day (using dictionary)

# Dictionary to map each day to a tally
fav_day_tallies = {
    "sat": 0,
    "sun": 0 }

for i in range(3):
    # Ask user their favourite day
    print("What is your favorite day?")
    fav = input(": ").lower().strip()

    # Add to day's tally
    fav_day_tallies[fav] += 1

# Summary
for day in fav_day_tallies:
    tally = fav_day_tallies[day]
    print(f"{day:5} = {tally:>3}")
```

```python
# Find everyone's favourite day (using lists)

# Parallel arrays storing days and tallies
days = ["sat", "sun"]
fav_day_tallies = [0] * len(days)

for i in range(3):
    # Ask user their favourite day
    print("What is your favorite day?")
    fav = input(": ").lower().strip()

    # Add to day's tally
    for i in range(len(days)):
        if days[i] == fav:
            fav_day_tallies[i] += 1

# Summary
for i in range(len(days)):
    day = days[i]
    tally = fav_day_tallies[i]
    print(f"{day:5} = {tally:>3}")
```

Reduce Hard-coding

How do you use a dictionary to keep track of menu orders?

```
orders = {"fries": 0, "burger": 0}
item = input("Your order? ")
if item == "fries":
  orders["fries"] += 1
elif item == "burger":
  orders["burger"] += 1
```

DO NOT DO THIS

```
orders = {"fries": 0, "burger": 0}
item = input("Your order? ")
if item in orders:
  orders[item] += 1
```

Use the key directly!

DO THIS

Let's review some concepts

A dictionary is like a list, but stores data as **key-value pairs.**

Dictionary can store any type of value.

Use **in** to check if a certain key appears used in the dictionary.

Use **for** loop to iterate over keys.