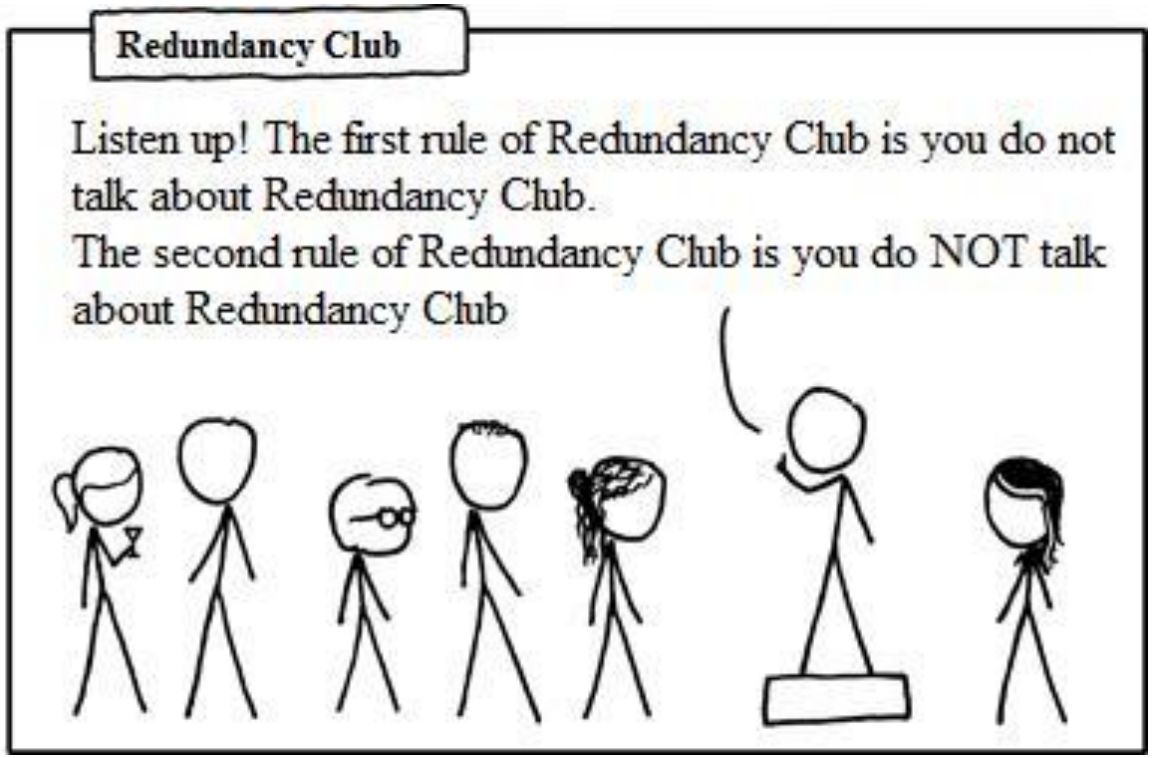
CSc 110, Autumn 2017

Lecture 6: Parameters

Adapted from slides by Marty Stepp and Stuart Reges



Redundant recipes

- Recipe for baking **20** cookies:
  - Mix the following ingredients in a bowl:
    - **4** cups flour
    - **1** cup butter
    - **1** cup sugar
    - **2** eggs
    - **40** oz. chocolate chips ...
  - Place on sheet and Bake for about 10 minutes.

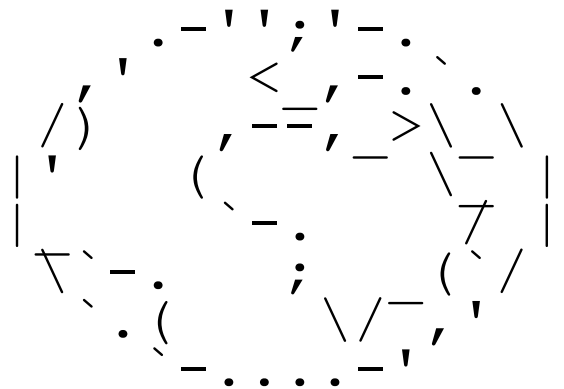
- Recipe for baking **40** cookies:
  - Mix the following ingredients in a bowl:
    - **8** cups flour
    - **2** cups butter
    - **2** cups sugar
    - **4** eggs
    - **80** oz. chocolate chips ...
  - Place on sheet and Bake for about 10 minutes.

Parameterized recipe

- Recipe for baking **20** cookies:
  - Mix the following ingredients in a bowl:
    - **4** cups flour
    - **1** cup sugar
    - **2** eggs
    - …
- Recipe for baking **N** cookies:
  - Mix the following ingredients in a bowl:
    - **N/5** cups flour
    - **N/20** cups butter
    - **N/20** cups sugar
    - **N/10** eggs
    - **2N** oz. chocolate chips ...
  - Place on sheet and Bake for about 10 minutes.
- **parameter**: A value that distinguishes similar tasks.

Redundant figures

- Consider the task of printing the following picture:



```
  /=__=\/T)
 /= = =\/
  ou ou
```

```
  /____\/T)
 /v v v\/
  ou ou
```

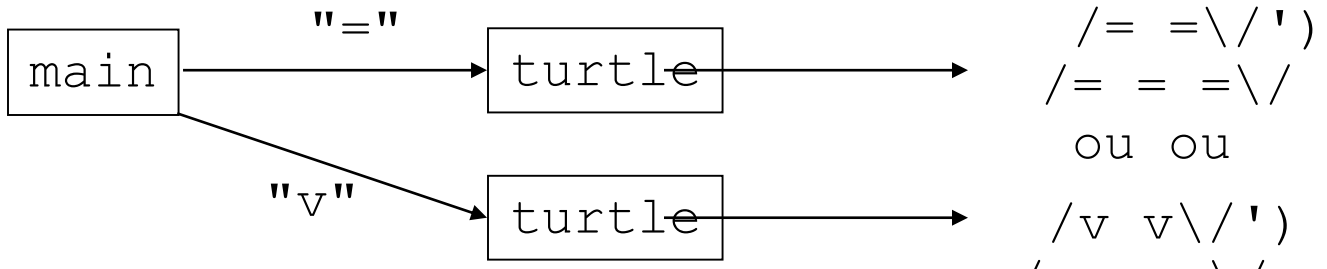
A redundant solution

```
def main():
    world()
    turtle_equal()
    turtle_v()

def world():
    print("     .-'';'-.")
    print("   ,'    <_,-.`.")
    print("  /)    ,--,_>\\_\\")
    print(" |'    (       \\_ |")
    print(" |       `-.     / |")
    print("  \`-.    ;    _(`/")
    print("   `.(     \/ ,'")
    print("     `-....-'")

def turtle_equal():
    print("            ____")
    print("       /== ==\\7')")
    print("      /= = =\\/")
    print("       ou ou")

def turtle_v():
    print("            ____")
    print("       /v v\\7')")
    print("      /v v v\\/")
    print("       ou ou")

main()
...
```

- This code is redundant.
- Would variables help? Would constants help?
- What is a better solution?
  - `turtle` - A function to draw a turtle of any shell pattern.

Parameterization

- **parameter**: A value passed to a function by its caller.
  - Instead of `turtle_equal`, `turtle_v`, write `turtle` to draw any turtle.
    - When *declaring* the function, we will state that it requires a parameter for the number of stars.
    - When *calling* the function, we will specify how many stars to draw.



Declaring a parameter

Stating that a function requires a parameter in order to run

```
def <name> (<name>):
    <statement>(s)
```

- Example:

```
def say_password(code):
    print("The password is:", code)
```

  - When `say_password` is called, the caller must specify the code to print.

Passing a parameter

Calling a function and specifying values for its parameters

<name> (***<expression>***)

- Example:

```
say_password(42)
say_password(12345)
```

Output:

```
The password is 42
The password is 12345
```

Parameters and loops

- A parameter can guide the number of repetitions of a loop.

```
chant(3)
```

```
def chant(times):
    for i in range(0, times):
        print("Just a salad...")
```

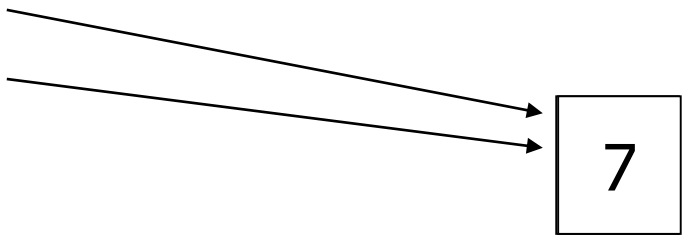
Output:

```
Just a salad...
Just a salad...
Just a salad...
```

How parameters are passed

- When the function is called:
  - The value is stored into the parameter variable.
  - The function's code executes using that value.

```
chant(3)
chant(7)
```




```
def chant(times):
    for i in range(0, times):
        print("Just a salad...")
```

Common errors

- If a function accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
chant()          # ERROR: parameter value required
```

- The value passed to a function must be of a type that will work.

```
chant(3.7)       # ERROR: must be of type int if it
                 #        is used as a range bound
```

- Exercise: Change the `counts` program to use a parameterized function for drawing lines of numbers.

Interactive programs

interactive program: Reads input from the console.

- While the program runs, it asks the user to type input.
- The input typed by the user is stored in variables in the code.

- Can be tricky; users are unpredictable and misbehave.
- But interactive programs have more interesting behavior.

`input`

- **`input`**: An function that can read input from the user.

- Using an `input` object to read console input:

  name `= input(`**prompt**`)`

  - Example:

    ```
    name = input("type your name: ")
    ```

    - The variable `name` will store the value the user typed in

input example

```
def main():
    age = input("How old are you? ")

    years = 65 - age
    print(years, "years until retirement!")
```

age | 29

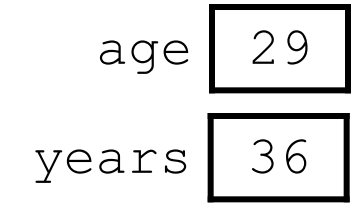
- Console (user input underlined):

```
How old are you? 29
```

```
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(65 - age)
TypeError: unsupported operand type(s) for -:
'int' and 'str'
```

input example

```
def main():
    age = int(input("How old are you? "))

    years = 65 - age
    print(years, "years until retirement!")
```



- Console (user input underlined):

```
How old are you? 29
36 years until retirement!
```