Access the slides and files here:

https://github.com/j-berg/bioinformatics_bootcamp

#3.2

Control flow

Modularizing tasks

Iterations

- How would you perform a similar task over and over for a list of variables?
- What if you don’t know how many times you want to repeat the task?

Conditions

- Helpful workflow logic

- if: the first option
- else if: the n^{th} option
- else: the final option

- Indentation matters in Python to help it know what tasks are part of which conditions

Conditions

- Say we want to get the output of a function and depending on that output, perform one of two options
- For example, we only want to run the next step if the output is less than 100. We will need if/else logic, and will need to use operators

```
y = function(x)


  if y < 100
     do a
   else
     do b
```

Conditions

- Let’s break this down and go over some edge cases

```
>>> if y < 100:
        do a
    else:
        do b
```

- Structure
  - The statement being evaluated follows “if” and ends with a “:”
  - The task below is run if the “if” statement is true
  - The syntax of how to do this will differ language to language
  - “else” covers all leftover options – this can be left out, or you can follow the ”else” with “pass”

```
else:
        pass
```

Operators

- >: greater than
- <: less than
- >=: greater than or equal to
- <=: less than or equal to
- ==: equal to
- !=: not equal to
- &: and (in Python, you can just use “and”)
- |: or (in Python, you can just use “or”)

https://www.statmethods.net/management/operators.html

- What if there are several options you want to encode?

```
> if y < 100:
     do this
  elif y > 100 and y <= 200:
     do this
  elif y > 200 and y <= 300:
     do this
  else:
     pass
```

For loops

- Used to repeat a task for a defined list of values

```
>>> A = [“Hello”, “world”, “!”]
>>> for x in A:
      print(x)
```

- This can be combined with conditions

```
>>> for x in A:
      if len(x) > 2:
            print(x)
```

- Thought exercise: What will this print out?

While loops

- What if we don’t know how many iterations we want to run?

```
>>> y = 0       # We initialize the counting variable here
>>> while y < 10:
      print(“Hello”)
      y += 1    # or y = y + 1
>>> print(”Goodbye”)
```

- However, this is flexible, if we changed the iterator to only add 0.5, would run 2x loops

Functions

- Helpful for packaging commonly used tasks
- Components:
  - Function name
  - Input variable(s)
  - Task
  - Output variable(s)

Functions

```
>>> def addone(x):
      x = x + 1
      return x
>>> addone(1)
2
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not define
```

Variables set or called in function are only available for that function call, will not set or modify outside variables (usually)

Functions

```
>>> def addone(x):
     x = x + 1
     return x
>>> x = addone(1)
>>> x
2
```

Setting the function to a variable saves the output

What about multiple inputs?

```
>>> def addtwo(x, y):
     z = (x + 1) / y
     return z
>>> addtwo(1, 2)
1.0
```

What about multiple outputs?

```
>>> def multitask(x):
        if x > 100:
                x = x / 2
                y = “large number”
        else:
                # we don’t need to do anything with x here as it is already defined
                y = “small number”
        return x, y

>> multitask(1)
(1, 'small number')
>>> x, y = multitask(101)
>>> print(y)
‘large number’
```

Homework

- Create a list of strings. For each string, if the number of characters is greater than 5, print out the string as is. If it is less than or equal to 5, print out the string plus “(this is a small word)”.
- Create a list of numbers. If the number is less than 100, divide by 2 and append to a new list. If the number falls between 100 and 200, divide by 3 and append to the new list. If the number is greater than 200, divide by 4 and append to the new list.

Important: For each task, create a function that takes as input the array to be processed and outputs the new array of modified values.